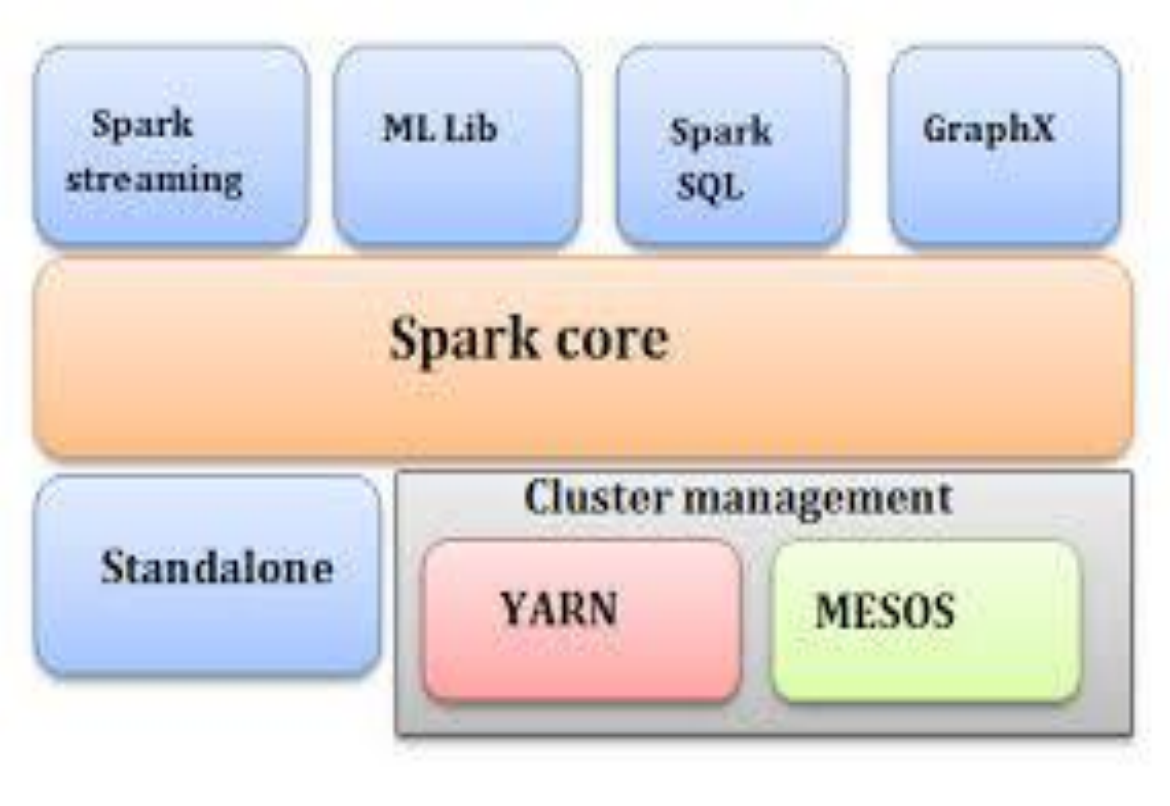
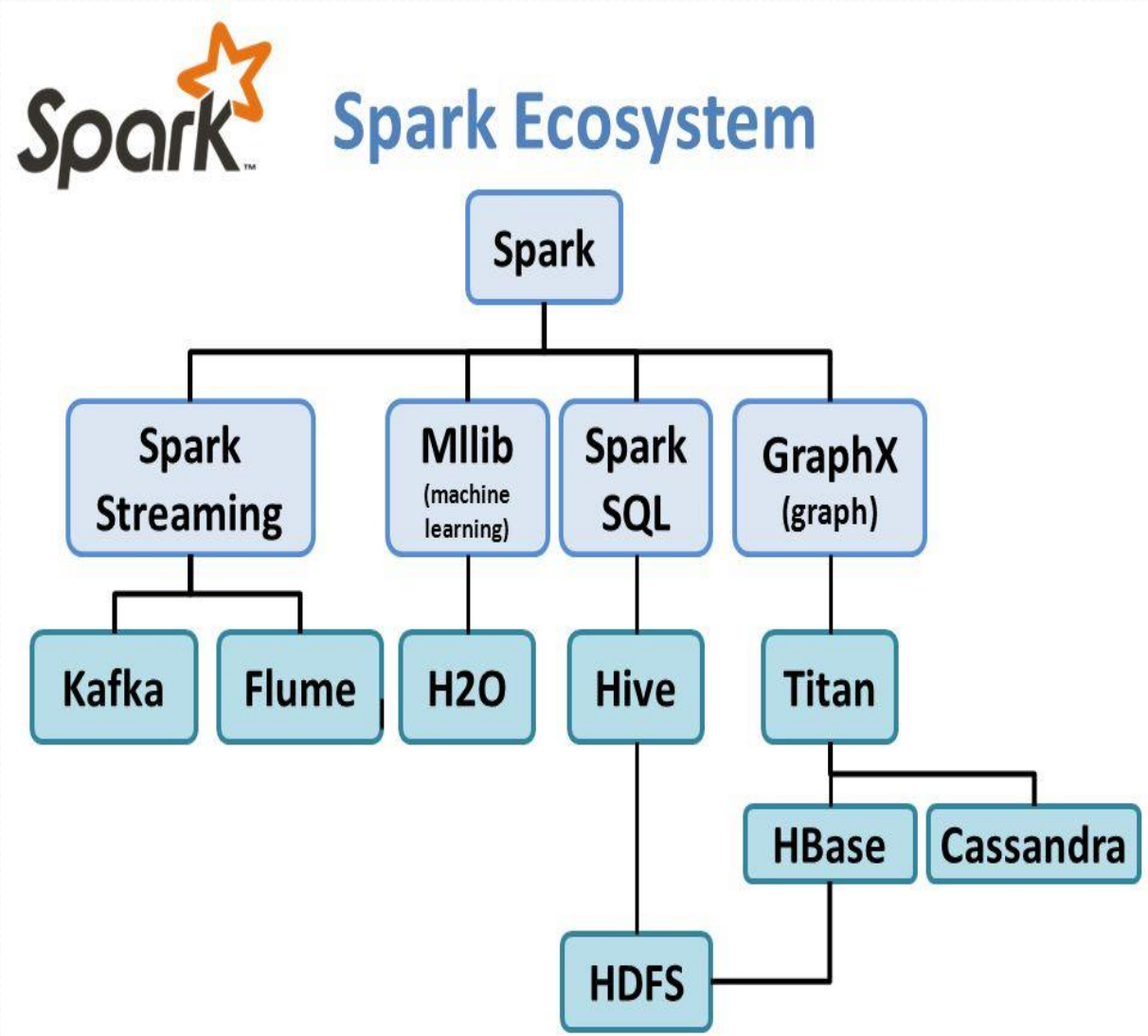
Apache Spark

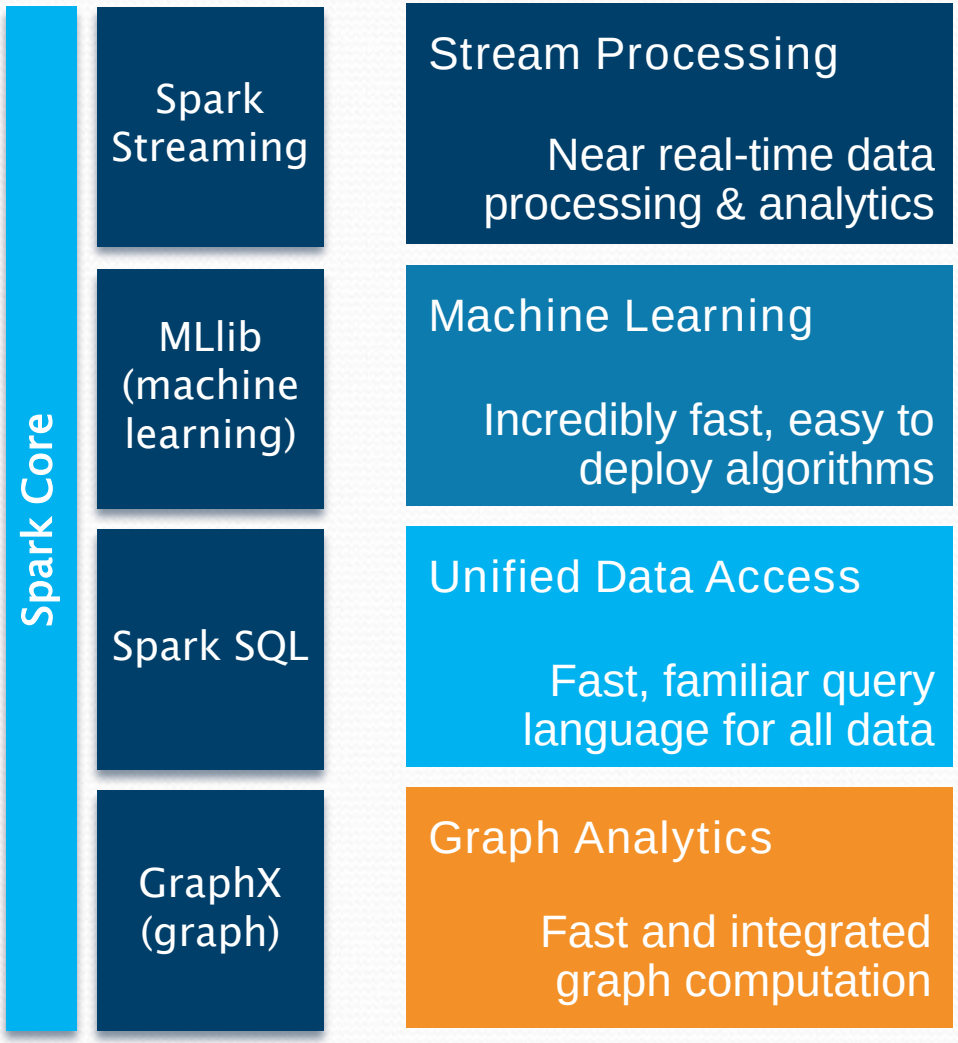
The Spark stack





Source: Mike Frampton (2015), Mastering Apache Spark, Packt Publishing

Spark Capabilities



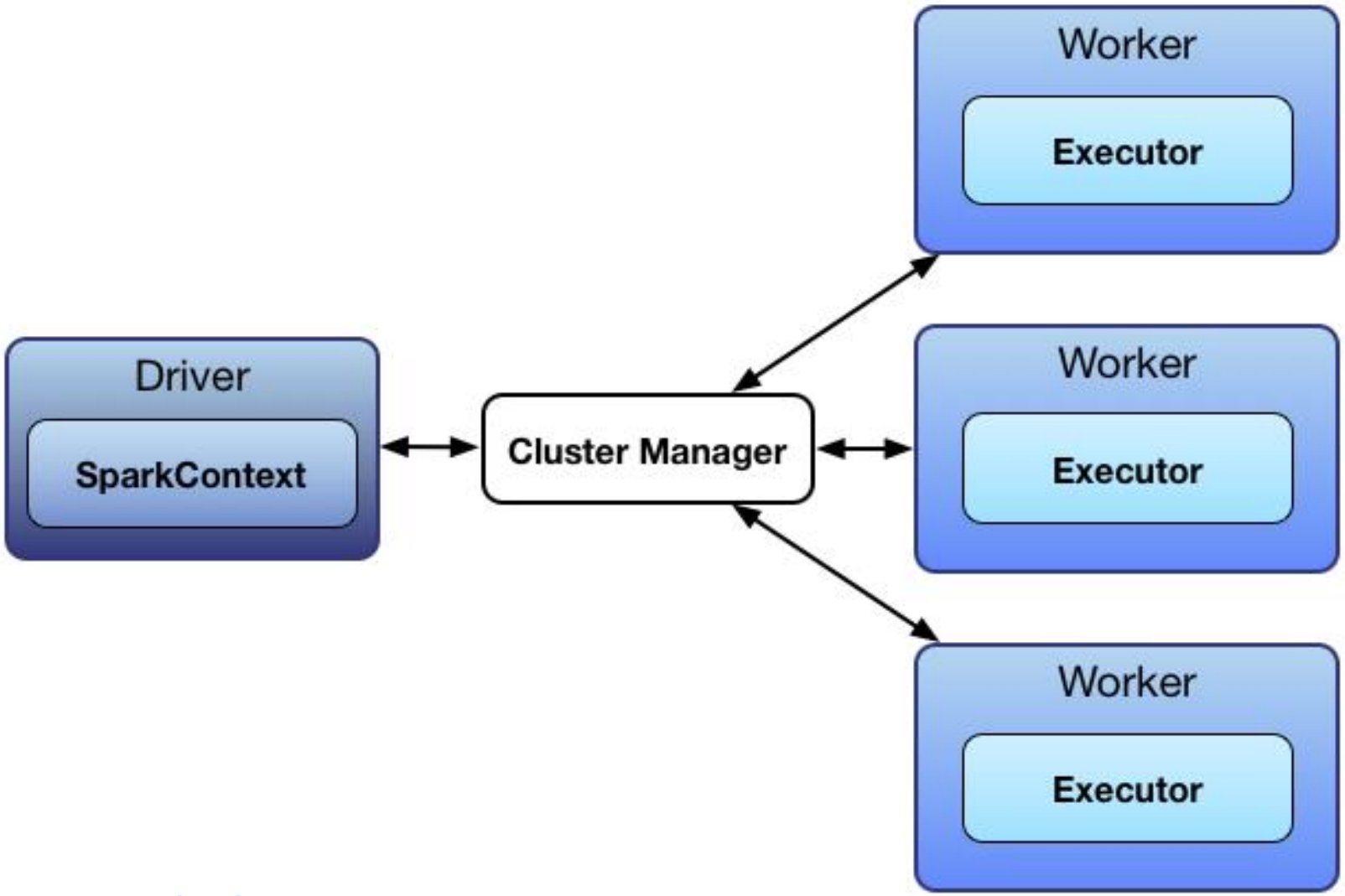
- Micro-batch event processing for near real-time analytics
- Process live streams of data (IoT, Twitter, Kafka)
- No multi-threading or parallel processing required

- Predictive and prescriptive analytics, and smart application design, from statistical and algorithmic models
- Algorithms are pre-built

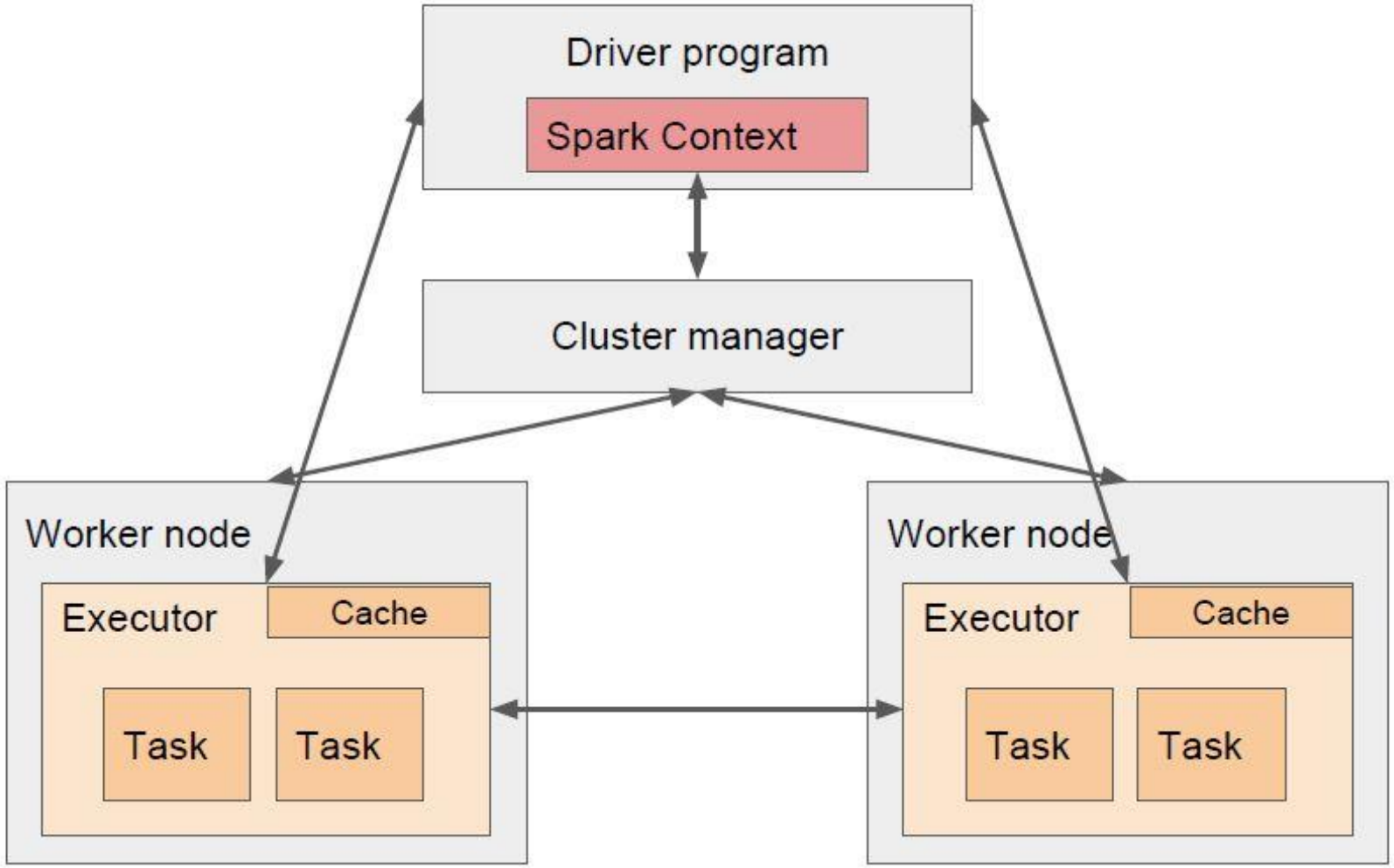
- Query your structured data sets with SQL or other dataframe APIs
- Data mining, BI, and insight discovery
- Get results faster due to performance

- Represent data in a graph
- Represent/analyze systems represented by nodes and interconnections between them
- Transportation, person to person relationships, etc.

Spark Architecture

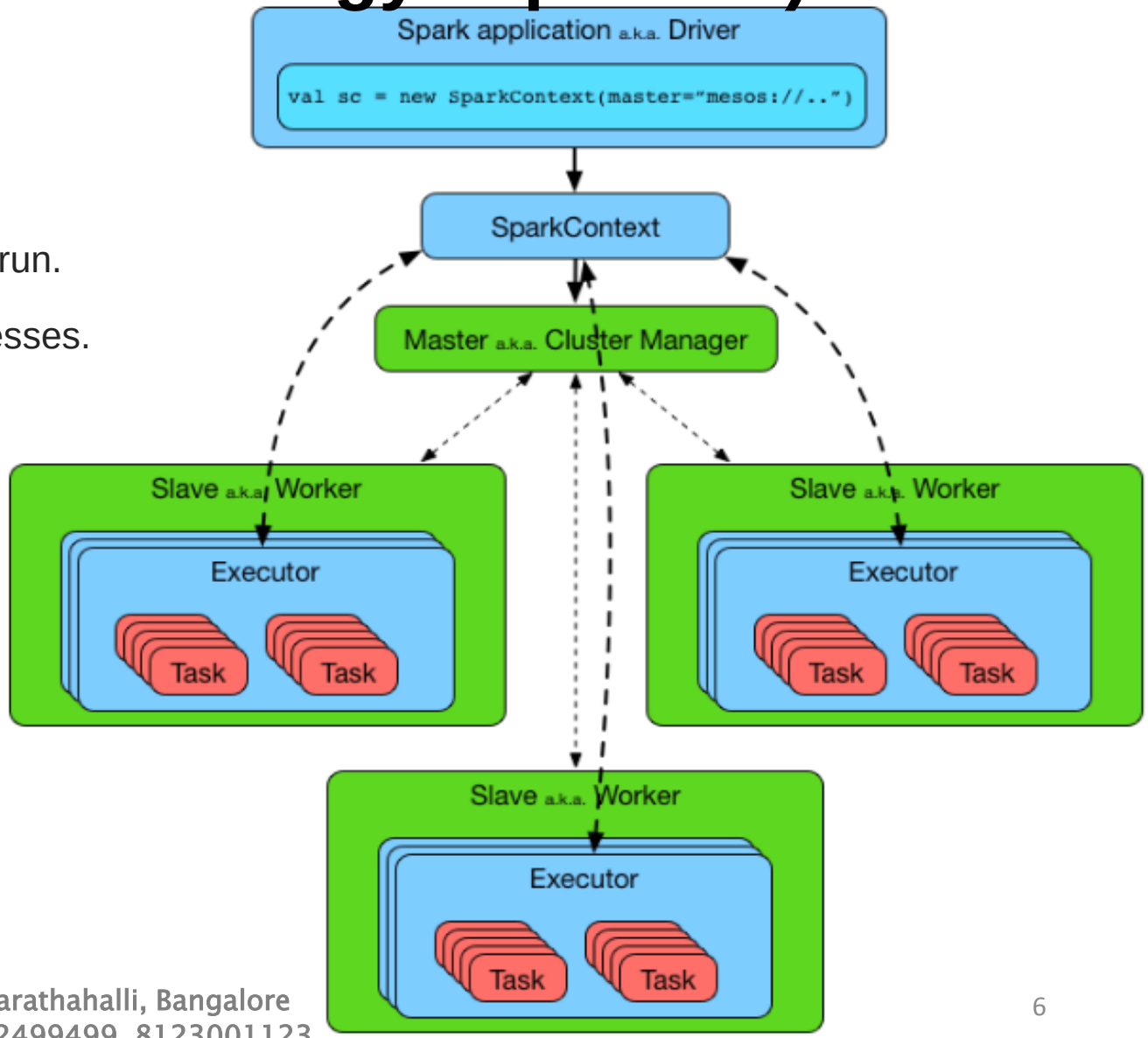


Spark Architecture



Spark Architecture (Terminology explained)

- Spark uses a master/worker(slave) architecture.
- There is a *driver* that talks to a single coordinator called master that manages *workers* in which *executors* run.
- The driver and the executors run in their own Java processes.
- You can run them all on the same (horizontal cluster) or separate machines (vertical cluster) or in a mixed machine configuration.



Spark Architecture - Driver

- Driver (Coordinator agent): A spark driver (aka an application's driver process) is a JVM process that hosts *SparkContext* for a Spark application.
- It is the cockpit of jobs and tasks execution (using *DAGScheduler* and *Task Scheduler*). It hosts Web UI for the environment.
- Simply stated, Driver is a program or process running the Spark context. Responsible for managing the job flow and scheduling tasks that will run on the executors.
- It splits a Spark application into tasks and schedules them to run on executors.
- A driver is where the task scheduler lives and spawns tasks across workers.
- A driver coordinates workers and overall execution of tasks.

- Context (Connection): Represents a connection to the Spark cluster. The Application which initiated the context can submit one or several jobs, sequentially or in parallel, batch or interactively, or long running server continuously serving requests.

Spark Architecture - Driver

- High-level control flow of work
- Your spark application runs as long as the Spark driver.
- - Once the driver terminates, so does your Spark application.
- Creates *SparkContext*, RDD's, and executes transformations and actions
- Launches *tasks*

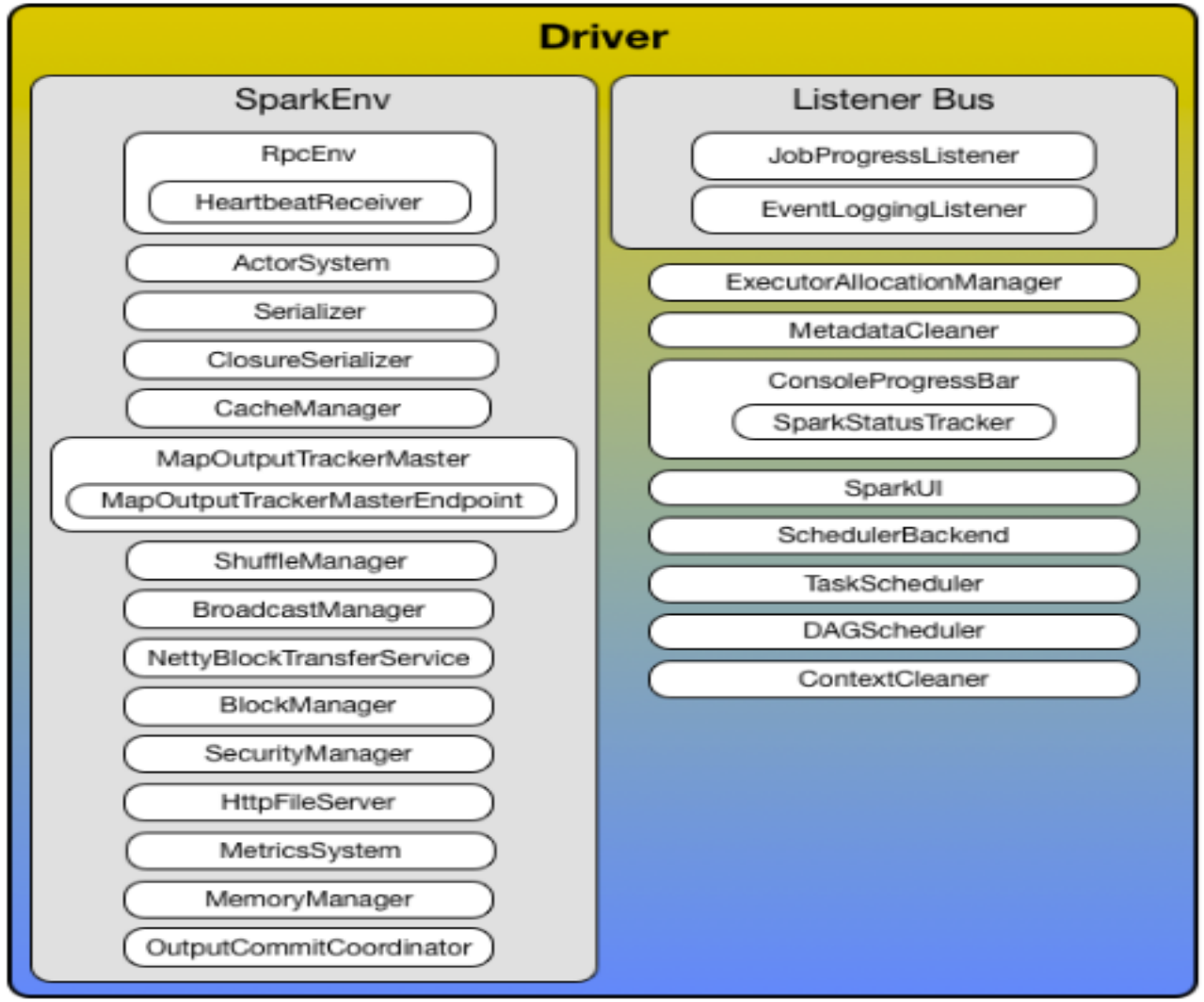


Figure 1.1 Driver with the services

Note: *Spark shell* is a Spark application and the driver. It creates a *SparkContext* that is available as ***sc***.

Spark Architecture

- Executors are distributed agents that execute tasks.
- Executors (Sub agent) are processes that run computation/task on a worker node and store data for a Spark application.
- They typically run for the entire lifetime of a Spark application and is called *static allocation of executors.*
- Executors send active task metrics to the driver and inform *executor backends* about task status updates (including task results).
- Executors provide in-memory storage for RDDs that are cached in Spark applications (via *Block Manager*)
- When executors are started they register themselves with the driver and communicate directly to execute tasks.
- Executors can run multiple tasks over its lifetime, both in parallel and sequentially. They track running tasks (by their task ids in *running Tasks* internal registry)
- Executors use a thread pool for launching tasks and sending metrics.
- Executors are managed exclusively by ***executor backends***.

Spark Architecture

- **Executor Backends** – Executor Backend is a pluggable interface used by executors to send status updates about the different states of a task to a scheduler.
- It is effectively a bridge between the driver and an executor, i.e. there are two endpoints running.
- At startup, an executor backend connects to the driver and creates an executor. It then launches and kills tasks. It stops when the driver orders so.

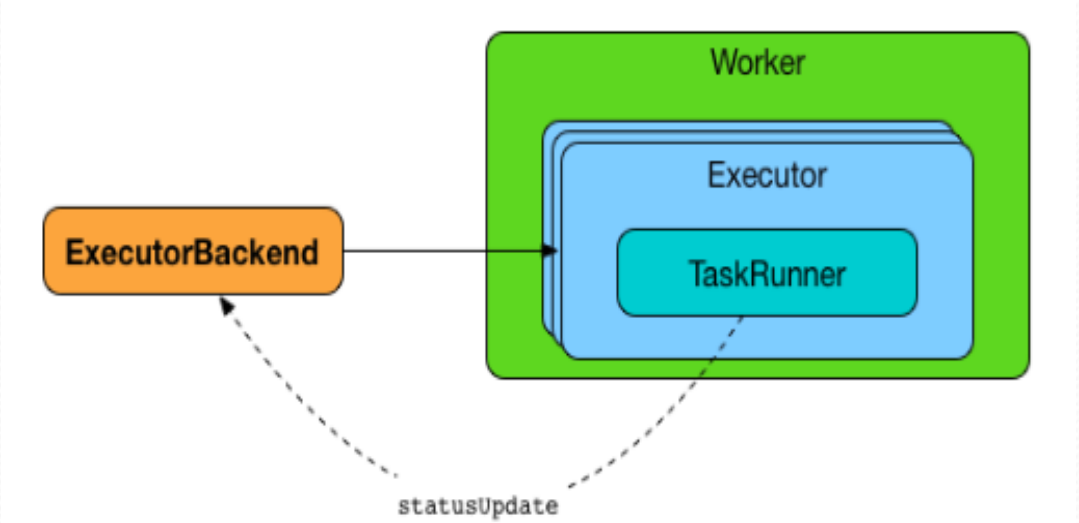


Figure 1. ExecutorBackends work on executors and communicate with driver

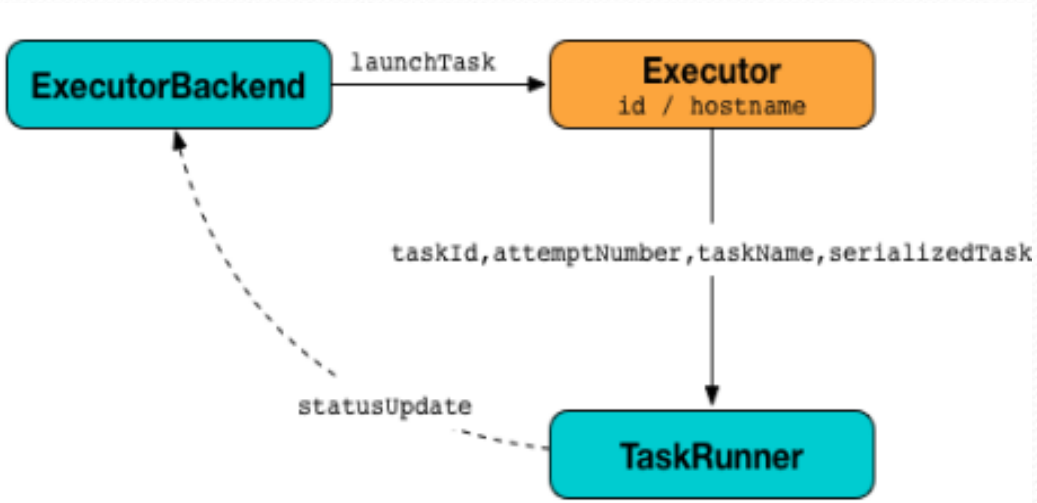


Figure 1. Launching tasks on executor using TaskRunners

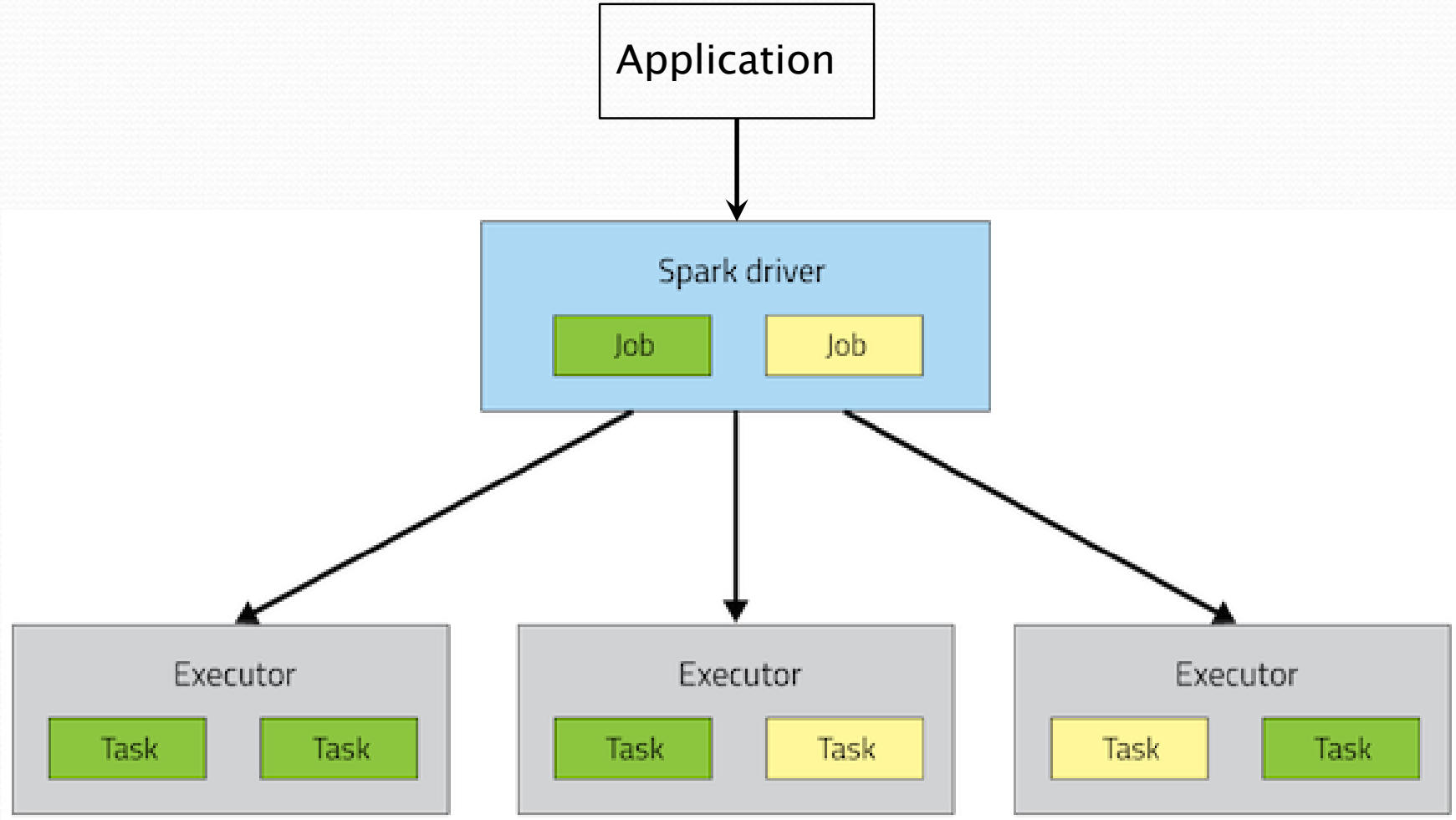
Spark Architecture

- **Block Manager** is a BlockDataManager, i.e. manages the storage for blocks that can represent cached RDD partitions, intermediate shuffle outputs, broadcasts, etc. It is also a BlockEvictionHandler that drops a block from memory and storing it on a disk if applicable.
- **Cluster Manager** is responsible for starting executor processes and where and when they will be run. Spark supports pluggable cluster manager, it supports YARN, Mesos and "standalone" cluster manager.
- **Workers (aka slaves)** are running Spark instances where executors live to execute tasks. They are the compute nodes in Spark.
  - A worker receives serialized tasks that it runs in a thread pool.
  - It hosts a local *Block Manager* that serves blocks to other workers in a Spark cluster. Workers communicate among themselves using their Block Manager instances.

Spark Architecture (Terminology)

- Job (Query / Query plan): A piece of logic (code) which will take some input from HDFS (or the local filesystem), perform some computations (transformations and actions) and write some output back.
- Stage (Sub plan): Jobs are divided into stages.
- Tasks (Sub section): Each stage is made up of tasks. One task per partition. One task is executed on one partition (of data) by one executor.

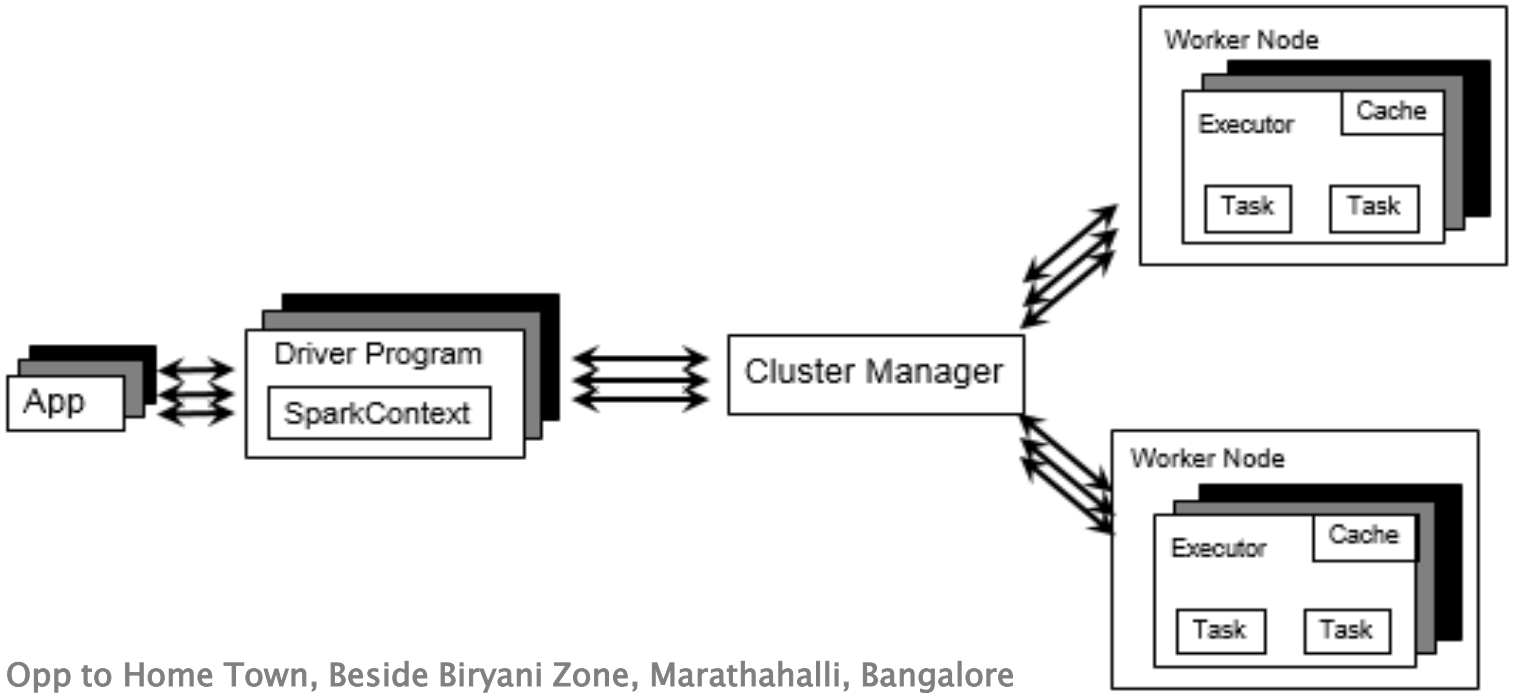
Putting the Spark concepts into a visual graph



Showing Multiple Apps

- Each Spark application runs as a set of processes coordinated by the Spark context object (driver program)
  - Spark context connects to Cluster Manager (standalone, Mesos/Yarn)
  - Spark context acquires executors (JVM instance)

    on worker nodes
  - Spark context sends tasks to the executors

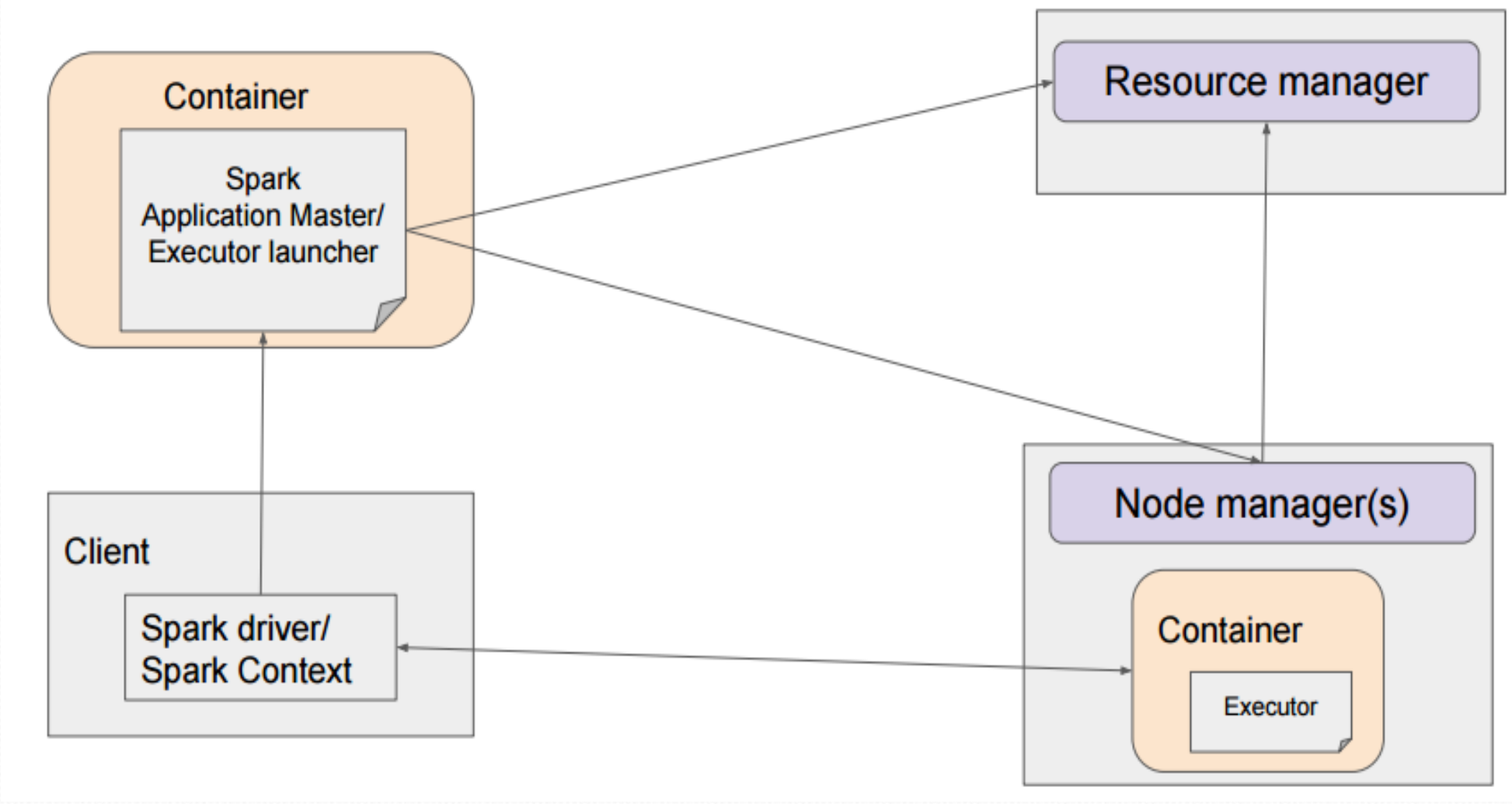


Modes of Spark on YARN

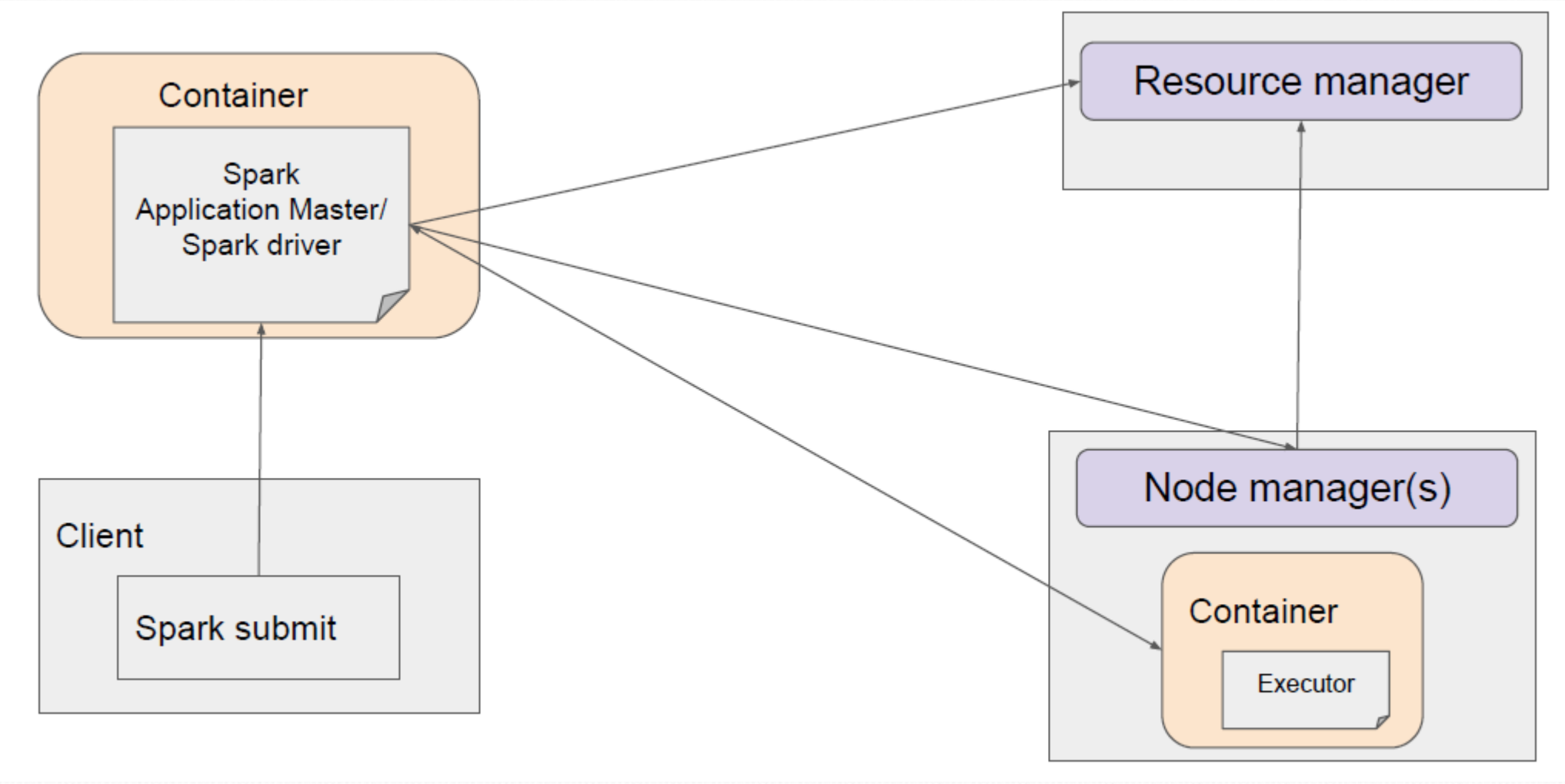
- **YARN-Client Mode**
  - Driver runs in the client process, and the application master is only used for requesting resources from YARN.
  - Used for interactive and debugging uses where you want to see your application's output immediately (on the client process side)

- **YARN-Cluster Mode**
  - In Yarn-cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application.
  - Yarn-cluster mode makes sense for production jobs.

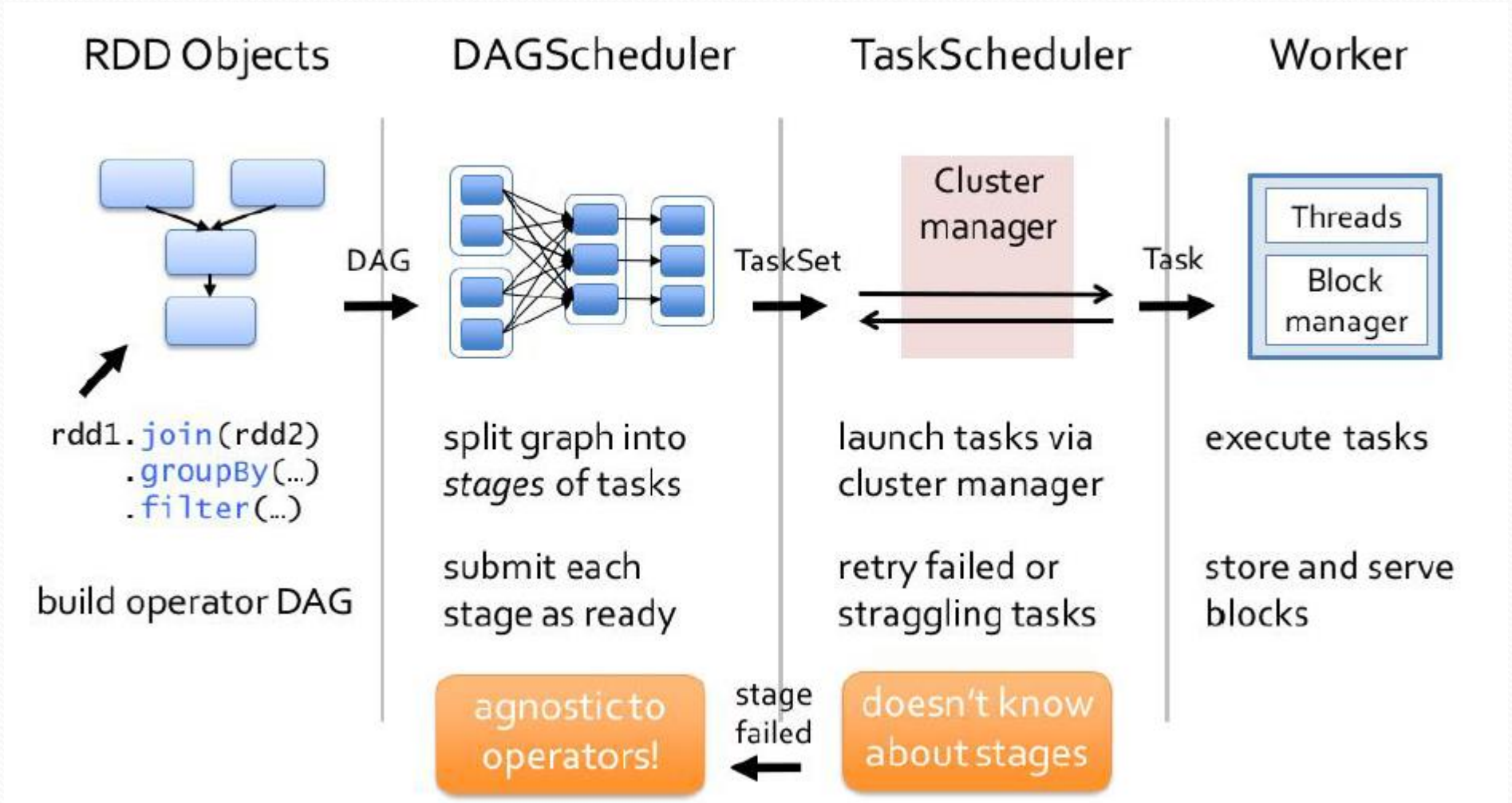
Spark in YARN client mode



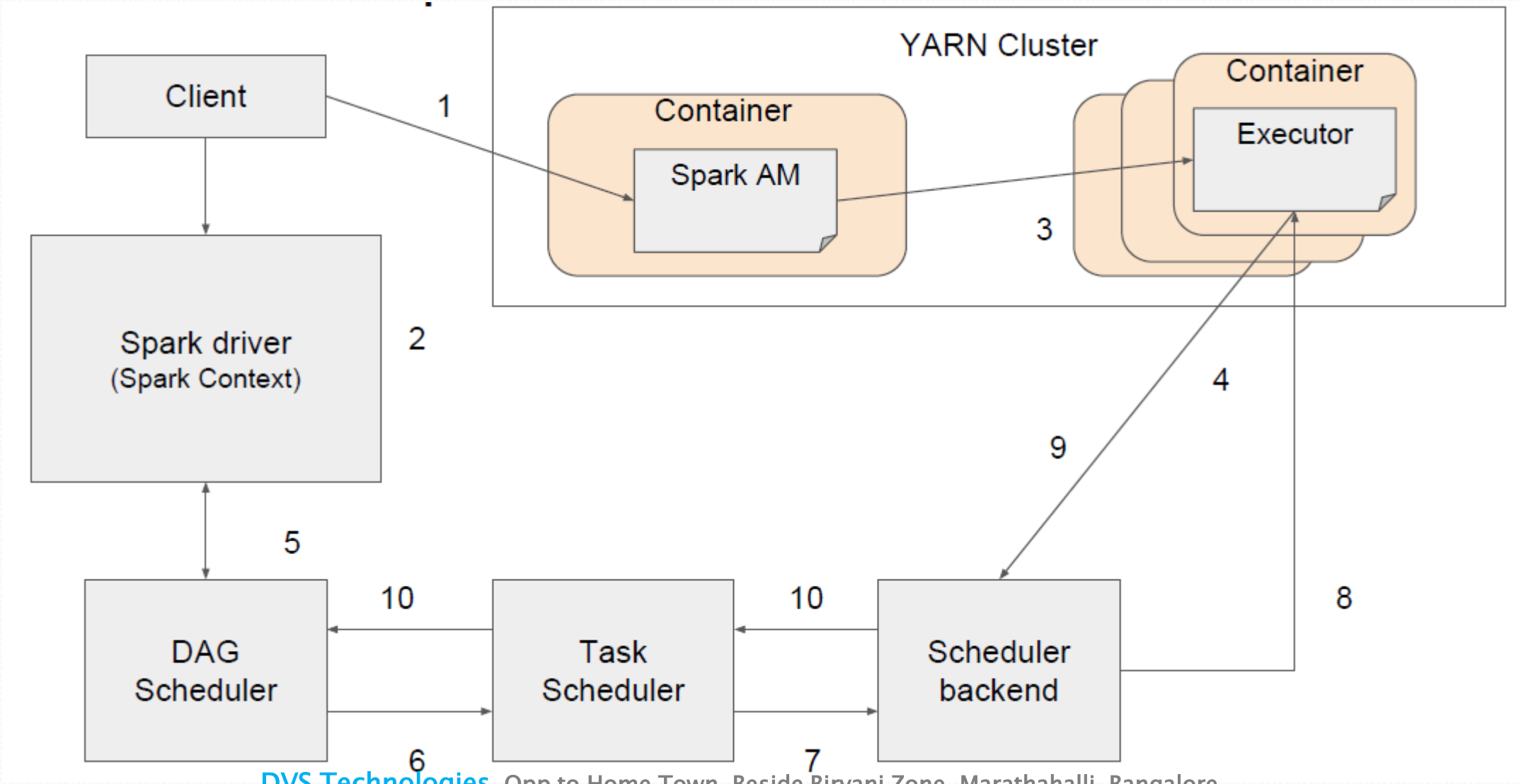
Spark in YARN cluster mode



Internals of Spark



Internals of Spark on YARN



Internals of Spark on YARN

1) Requests container for the AM and launches AM in the container.
2) Creates SparkContext (inside AM / inside Client). This internally creates a DAG Scheduler, Task scheduler and Scheduler backend.
3) Application master based on the required resources will request for the containers. Once it get the containers it runs executor process in the container.
4) The executor process when it comes up register with the Scheduler backend.
5) When few lines of code has to be run on the cluster, RDD runJob method calls the DAG scheduler to create a DAG of tasks.

Internals of Spark on YARN

6) Set of tasks which is capable of running in parallel is sent to the Task Scheduler in the form of TaskSet.

7) Task scheduler in turn will contact the Scheduler backend to run the tasks on the executor.

8) Scheduler backend which keeps track of running executors and its statuses, will schedule tasks on executors.

9) Task output if any are sent through heartbeats to Scheduler backend

10) Scheduler backend passes the task output onto the Task and DAG scheduler which could make use of that output.