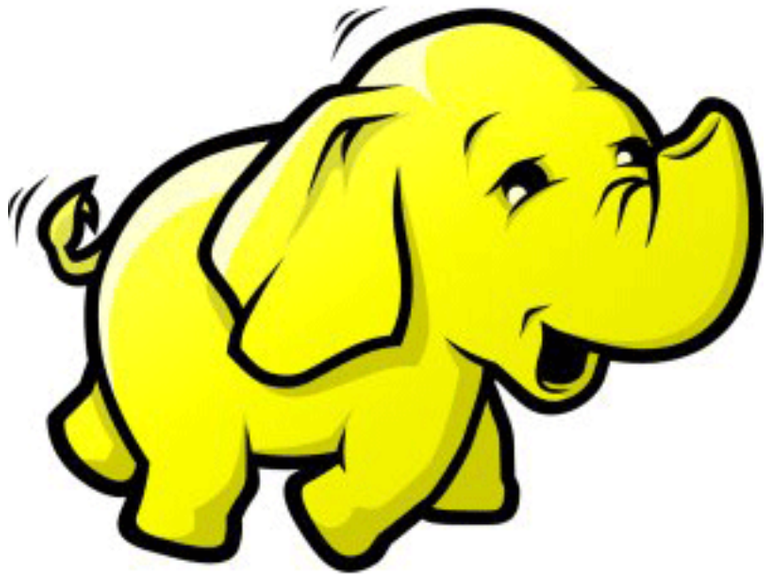
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
ⵜⴰⵖⵉⵔⵉⵜ ⵜⴰⵎⴰⵡⴰⵙⵜ ⴻⵏ ⵡⵎⴰⵙⵙ

جامعة سيدي محمد بن عبدالله
كلية العلوم ظهر المهراز

Université Sidi Mohamed Ben Abdellah
Faculté des Sciences Dhar Mahraz



MapReduce V2 & YARN

Présenté par :

Anouar Joual , Sadiqi Omar , Salah Eddine Ramah , Manariyo Daniel , Imane Echchahedy el ouazzany

Encadré par : Pr Chahhou Mohammed

PLAN

1. Introduction
2. MapReduce V1 : Into the code
3. MapReduce V2 Vs Yarn : Wrong comparaison
4. YARN
5. Implementation
6. Conclusion

The main Actors :

The Client

The JobTracker

The TaskTrackers

HDFS

Job submission Scenario :

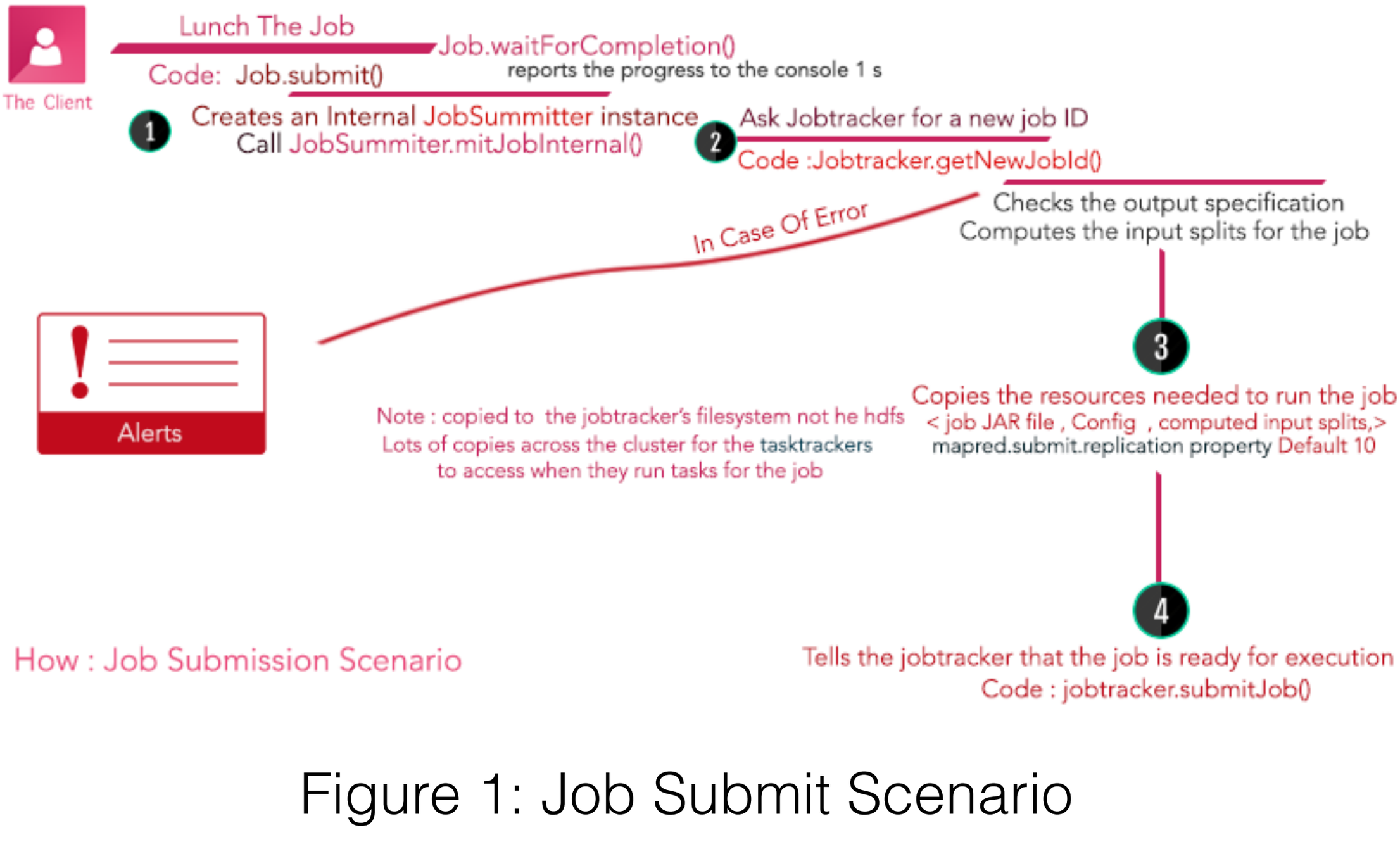


Figure 1: Job Submit Scenario

Job submission Scenario :

```
// Create a new Job
    Job job = Job.getInstance();
    job.setJarByClass(MyJob.class);

    // Specify various job-specific parameters
    job.setJobName("myjob");

    job.setInputPath(new Path("in"));
    job.setOutputPath(new Path("out"));

    job.setMapperClass(MyJob.MyMapper.class);
    job.setReducerClass(MyJob.MyReducer.class);

    // Submit the job, then poll for progress until the job is complete
    job.waitForCompletion(true);
```

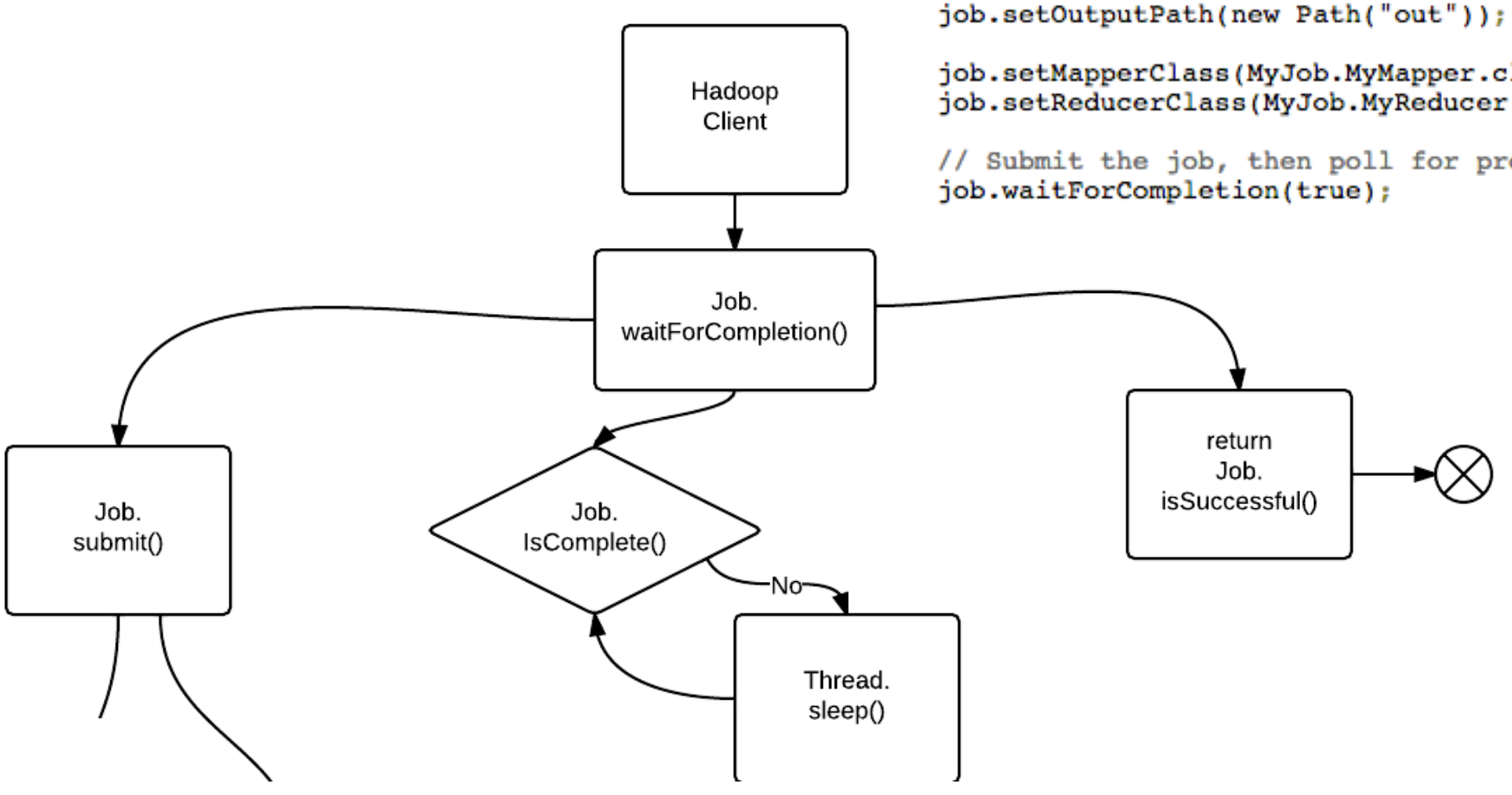


Figure 2 : Client Side

Job submission Scenario :

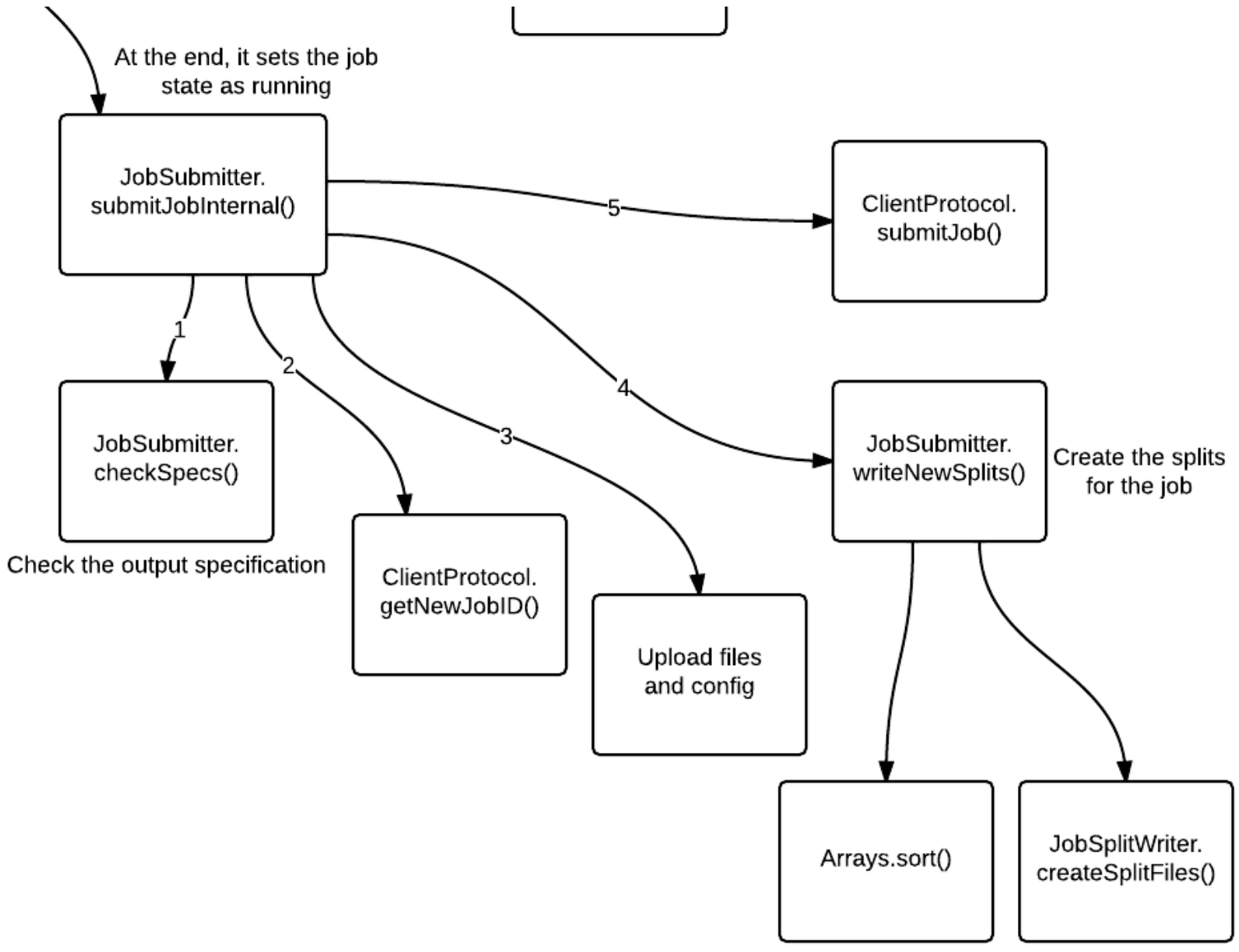


Figure 3: Internal Work

Job initialization

When the JobTracker receives a call to its submitJob() method, it puts it into an internal queue from where the job scheduler will pick it up and initialize it.

Initialization involves creating an object to represent the job being run, which encapsulates its tasks, and bookkeeping information to keep track of the status and progress of its tasks

Job initialization

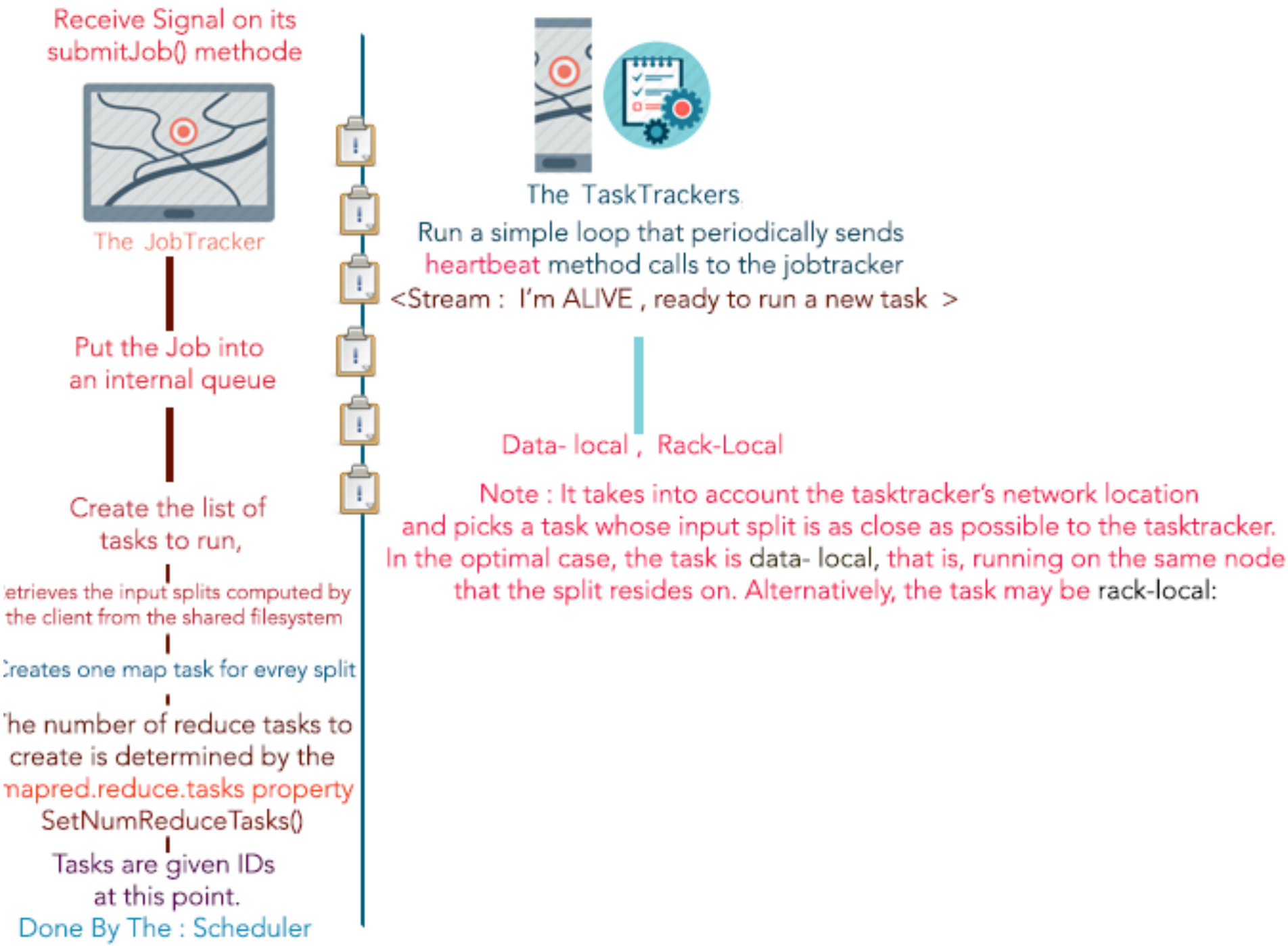


Figure 4: Job initialization

The tasktracker has been assigned a task :

Step 1 : copying All needed Files from the shared filesystem to the tasktracker's filesystem

Step 2 :Creates a local working directory for the task , creates an instance of taskRunner ()to run the task

Note : TaskRunner use separated JVM , so that any bugs in the user-defined map and reduce functions don't affect the tasktracker Communication through *umbilical interface*

Streaming and pipes

Both Streaming and Pipes run special map and reduce tasks for the purpose of launching the user-supplied executable and communicating with it

In the case of Streaming, the Streaming task communicates with the process (which may be written in any language) using standard input and output streams. The Pipes task,

on the other hand, listens on a socket and passes the C++ process a port number in its environment so that on startup, the C++ process can establish a persistent socket connection back to the parent Java Pipes task.

Progress and status updates

MapReduce jobs are long-running batch jobs

Status : Running, successfully completed, failed

For map tasks, this is the proportion of the input that has been processed.

dividing the total progress into three parts, corresponding to the three phases of the shuffle

For example, if the task has run the reducer on half its input, the task's progress is 5/6, since it has completed the copy and sort phases (1/3 each) and is halfway through the reduce phase (1/6).

Job completion

When the jobtracker receives a notification that the last task for a job is complete , it changes the status for the job to "successful." Then, when the Job polls for status, it learns that the job has completed successfully, so it prints a message to tell the user and then returns from the waitForCompletion() method.

Job statistics and counters are printed to the console at this point.

For very large clusters in the region of 4,000 nodes and higher, the MapReduce system described in the previous section begins to hit scalability bottlenecks,

so in 2010 a group at Yahoo! began to design the next generation of MapReduce. The result was YARN, short for Yet Another Resource Negotiator

YARN is more general than MapReduce, and in fact MapReduce is just one type of *YARN application*. There are a few other YARN applications, such as a distributed shell that can run a script on a set of nodes in the cluster, and others are actively being developed

MapReduce is Programming Model

YARN is architecture for distribution cluster

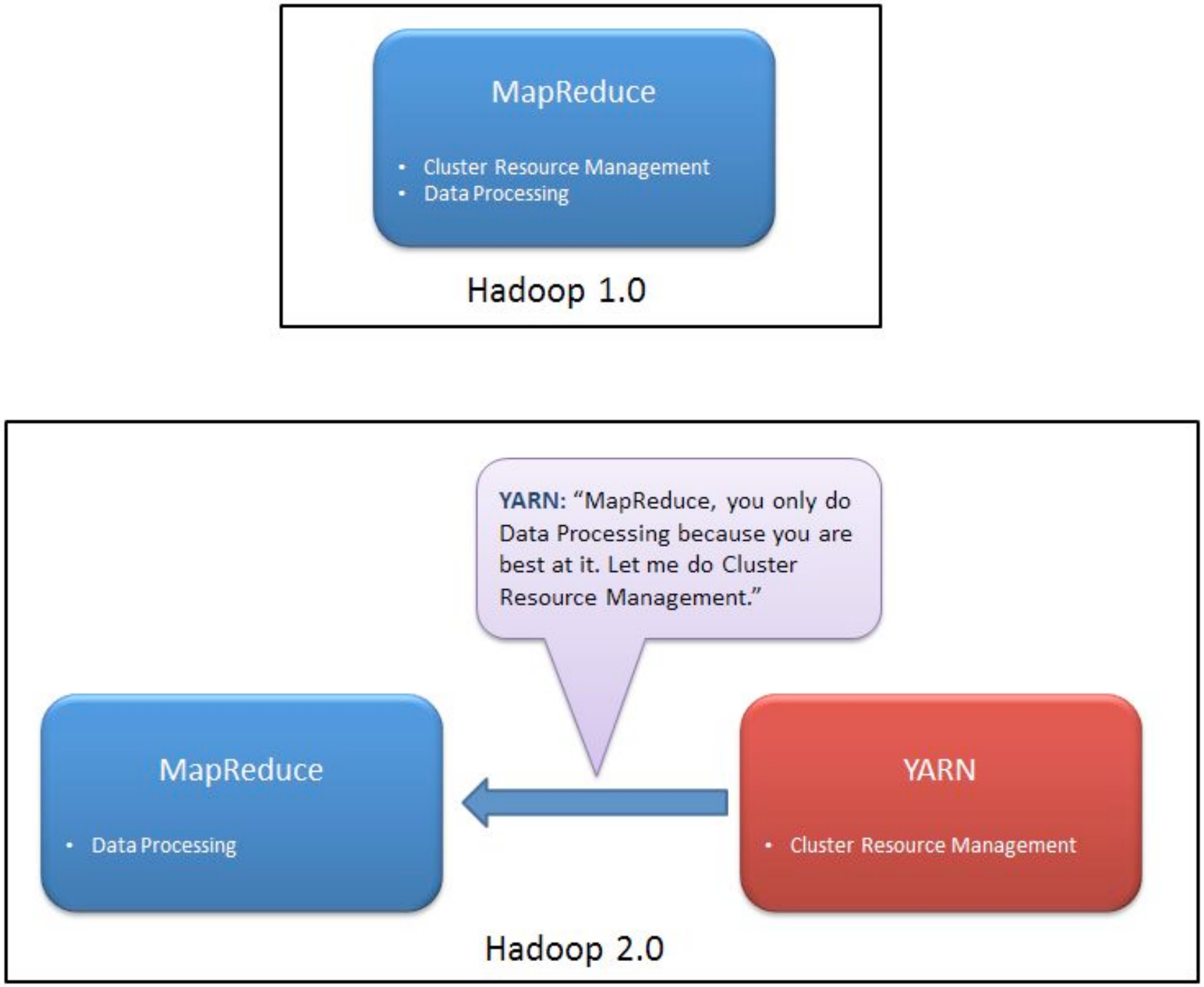


Figure 5 : Hadoop 1 & hadoop 2

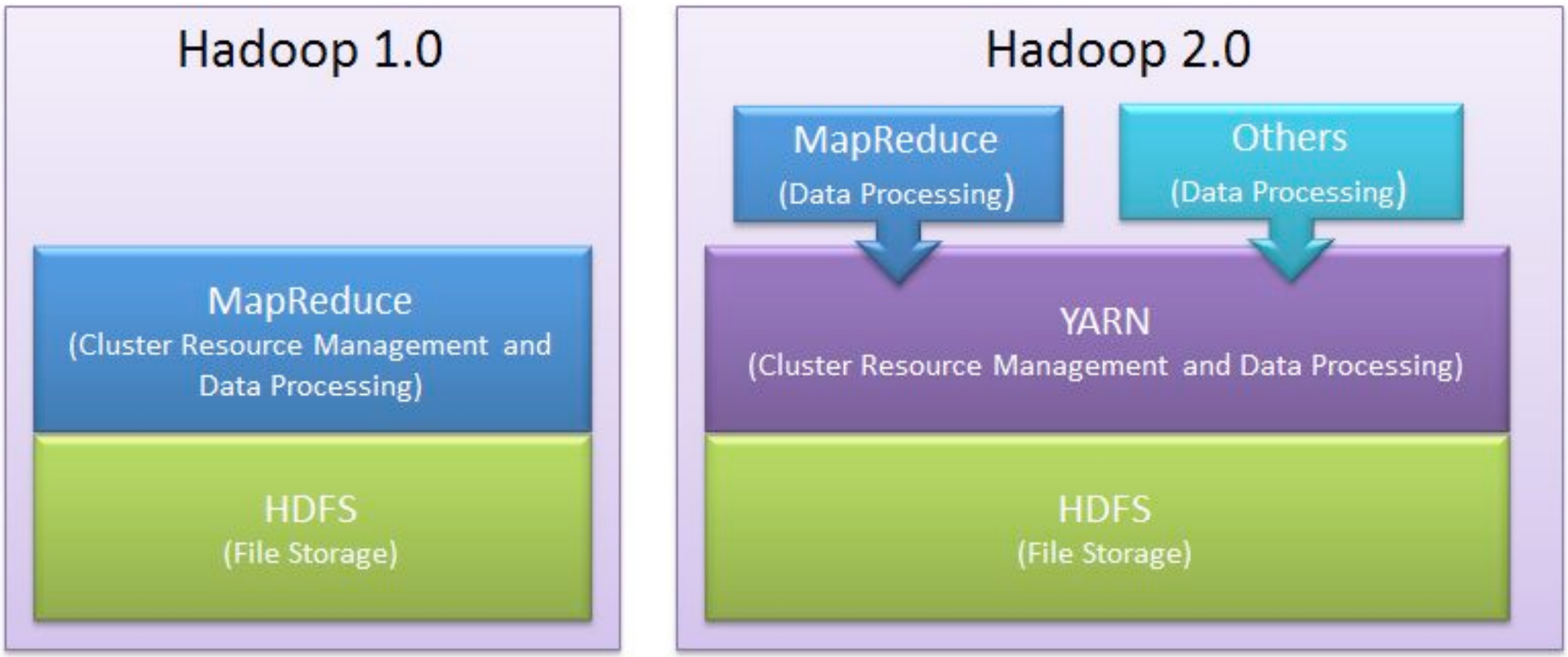


Figure 5 : Hadoop 1 & hadoop 2

The **jobtracker** takes care of both job scheduling and task progress monitoring.

YARN separates these two roles into two independent daemons: a resource manager to manage the use of resources across the cluster and an application master to manage the lifecycle of applications running on the cluster.

Job submission

Jobs are submitted in MapReduce 2 using the same user API as MapReduce 1 .
the job is submitted by calling submitApplication() on the resource manager

Job initialization

When the resource manager receives a call to its submitApplication(), it hands off the request to the scheduler. The scheduler allocates a container, and the resource manager then launches the application master's process there,

Job initialization

The application master for MapReduce jobs is a Java application It initializes the job by creating a number of bookkeeping objects to keep track of the job's progress, as it will receive progress and completion reports from the tasks

Nothing Special So far !!!!!!!!!

Job initialization

The next thing the application master does is decide how to run the tasks that make up the MapReduce job.

If the job is small, the application master may choose to run the tasks in the same JVM as itself

Cost Probleme : Running them in parallel, compared to running them sequentially on one node

Small : 10 Mappers , 1 Reducer , input size that is less than the size of one HDFS block : We say *uberized*, or run as an *uber task*.

Job initialization

The scheduler uses this information to make scheduling decisions. It attempts to place tasks on **data-local** nodes in the ideal case, but if this is not possible, the scheduler prefers **rack-local** placement to nonlocal placement.

information about each map task's data locality, in particular the hosts and corresponding racks that the input split resides on.

mapreduce.map.memory.mb &
mapreduce.reduce.memory.mb
Memory Allocated for each mapper or reducer

Job initialization

Problems Fixed :
underutilization when tasks use less memory (because other waiting tasks are not able to take advantage of the unused memory) and problems of job failure when a task can’t complete since it can’t get enough memory to run correctly and therefore can’t complete.

Task execution

Once a task has been assigned a container by the resource manager's scheduler, the application master starts the container by contacting the node manager

The YarnChild runs in a dedicated JVM, for the same reason that tasktrackers spawn new JVMs for tasks in MapReduce 1: to isolate user code from long-running system daemons. Unlike MapReduce 1, however, YARN does not support JVM reuse, so each task runs in a new JVM

Introduction | MapReduce V1 | Vs Yarn | YARN | Implementation