Resource and Application Management

Diego Rivera Correa

Slides owned by: Mirek Riedewald

Key Learning Goals

- What is the difference between cluster resource manager and application master?
- What is the driver of a job?
- What is the difference between a job and a task?
- What is the main reason for separating resource management from application management?

Introduction

- Cluster = A collection of networked machines (nodes) that work together as a unified system to store and process data.

- Clusters have resources; jobs and services need resources. The resource manager decides *who* gets *which* resources *when*.
  - Example: If 5 Spark jobs and 2 MapReduce jobs are submitted simultaneously, the resource manager decides which machines each one gets.

Introduction

- There is a **single resource manager** for the entire cluster. As we have seen for the distributed file system, this simplifies consensus in a distributed system.
  - And like the GFS master process, the resource manager only exchanges small control messages with processes requesting resources.

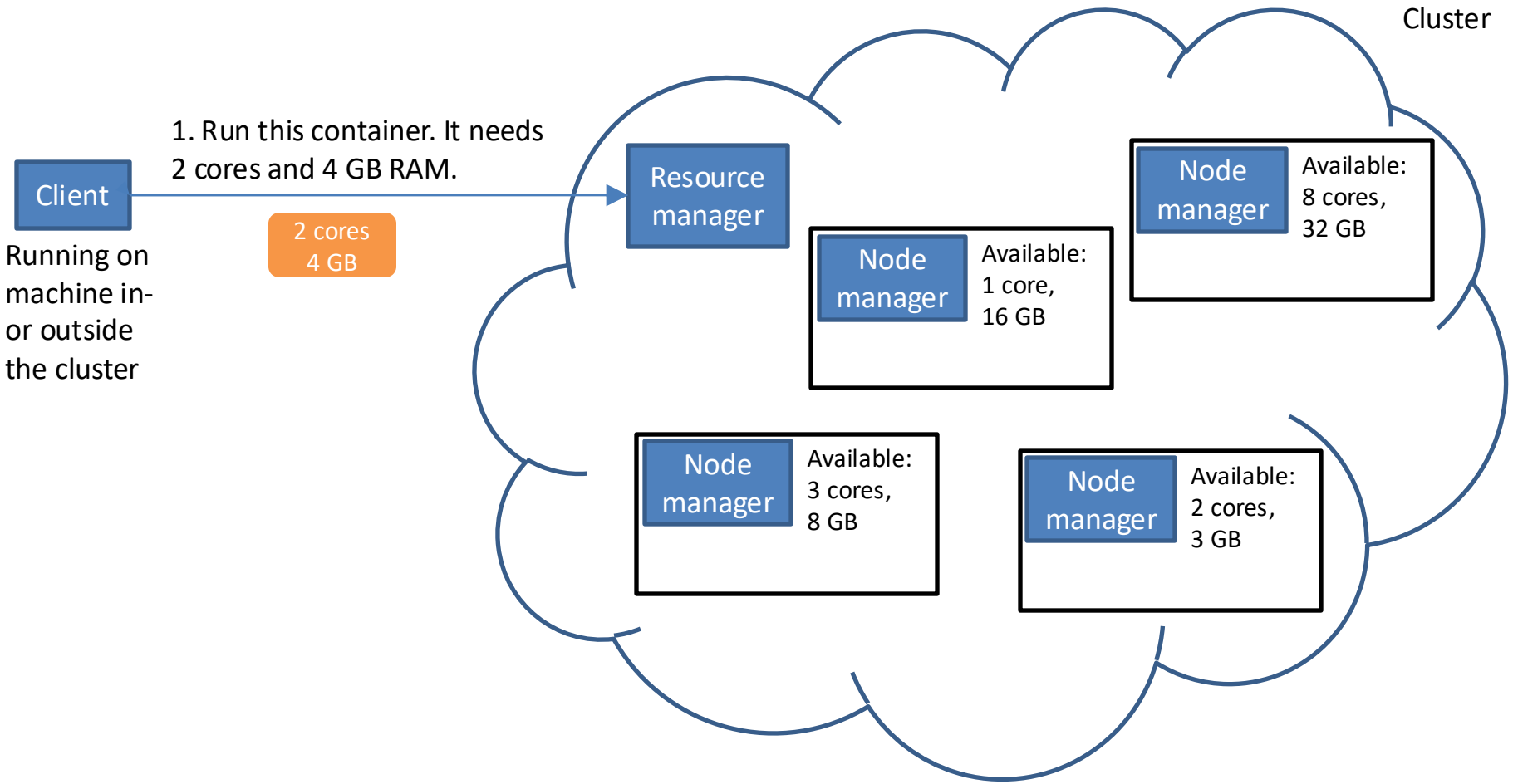
- Imagine if every node independently decided who gets resources — conflicting allocations would be constant!

Introduction

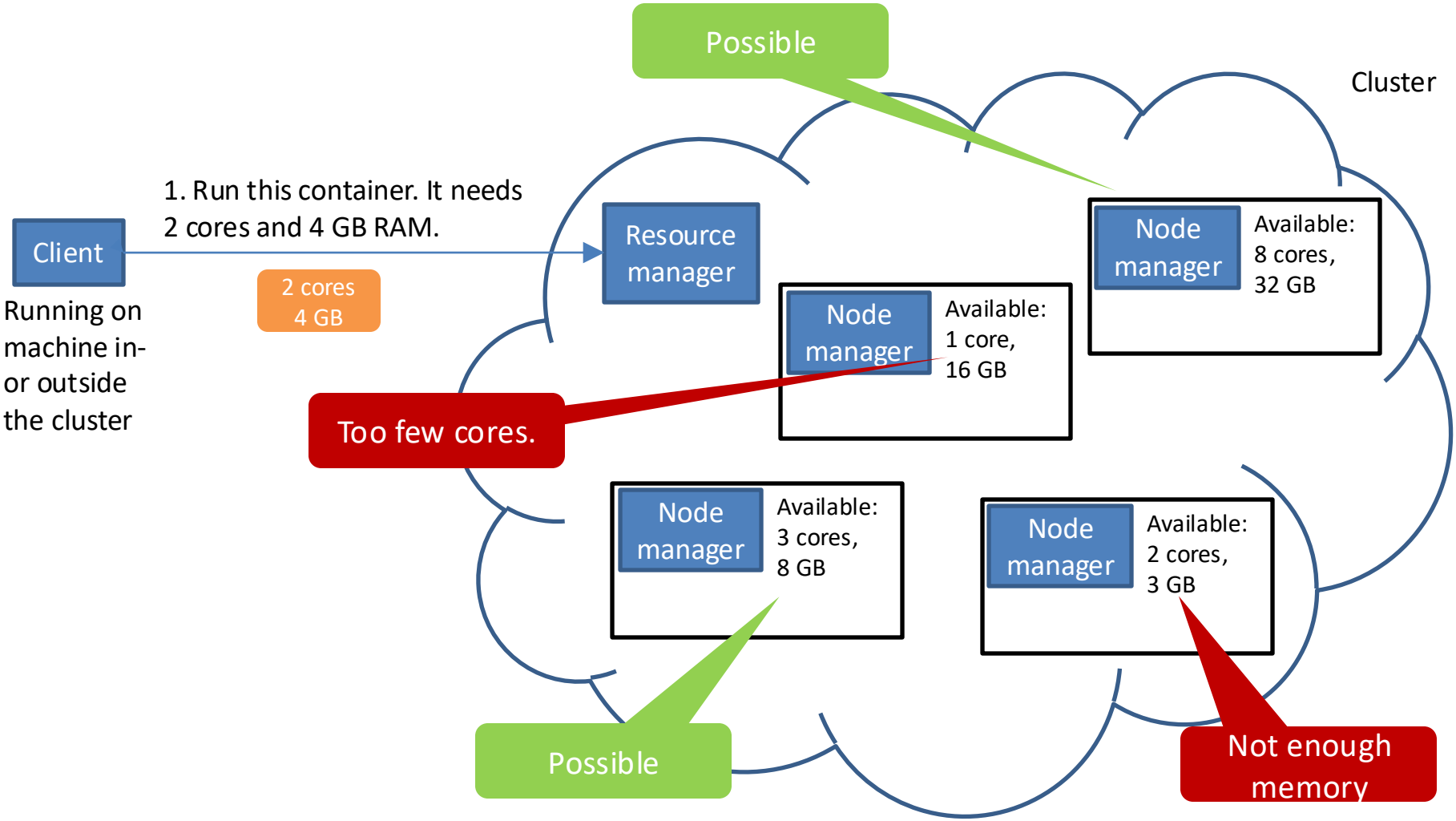
- In addition to MapReduce and Spark, many other services and applications may run in a cluster. They all communicate with the same cluster resource manager.

- Scheduling is a classic computer-science problem with many solutions. We focus on the functionality of YARN, a popular open-source scheduler.

Basic View of Cluster Resource Management



Who Will Run the Application?



Who Will Run the Application?

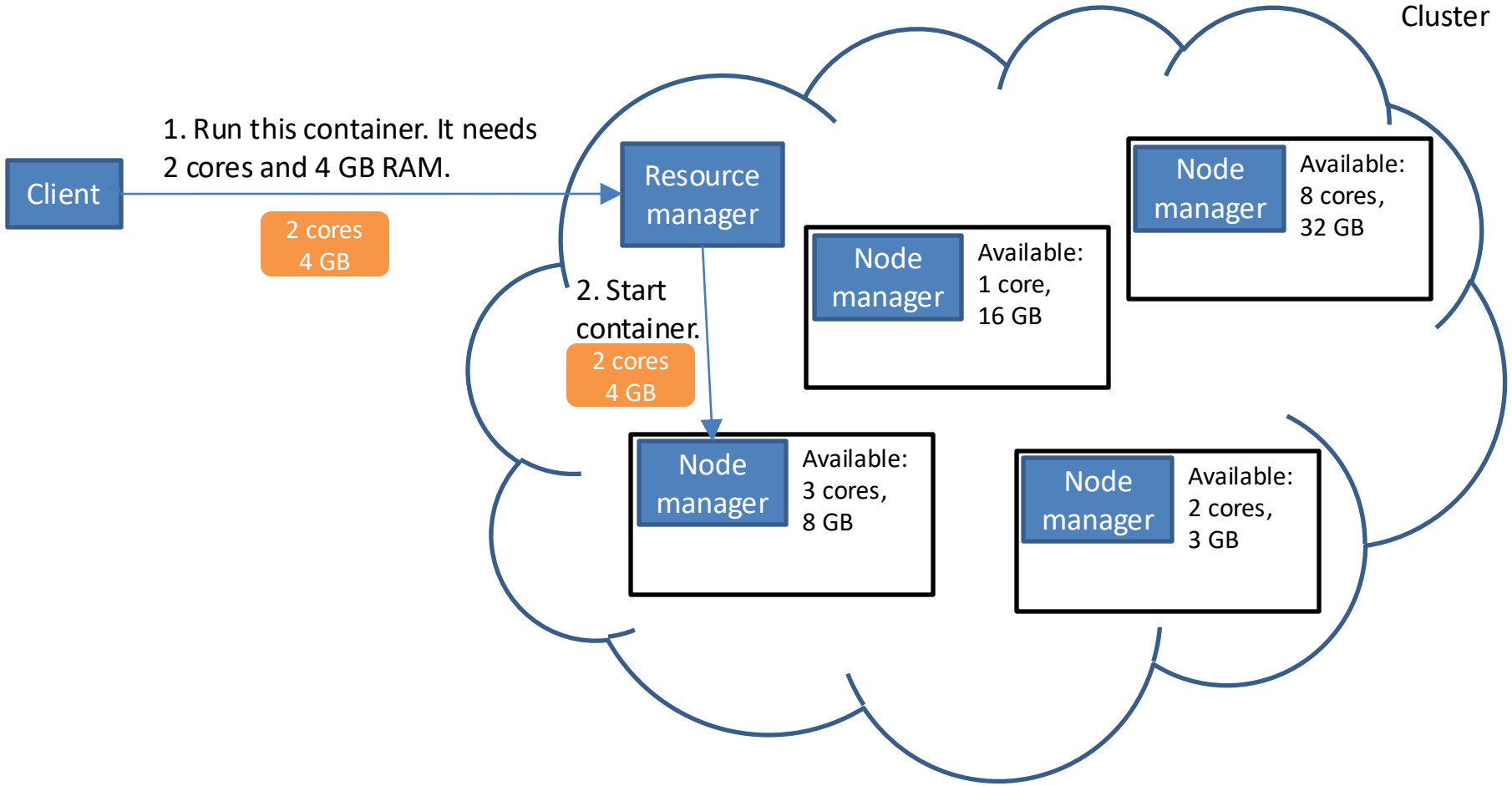
- The resource manager communicates with the node managers running on the machines to keep an up-to-date view about available resources.

Who Will Run the Application?

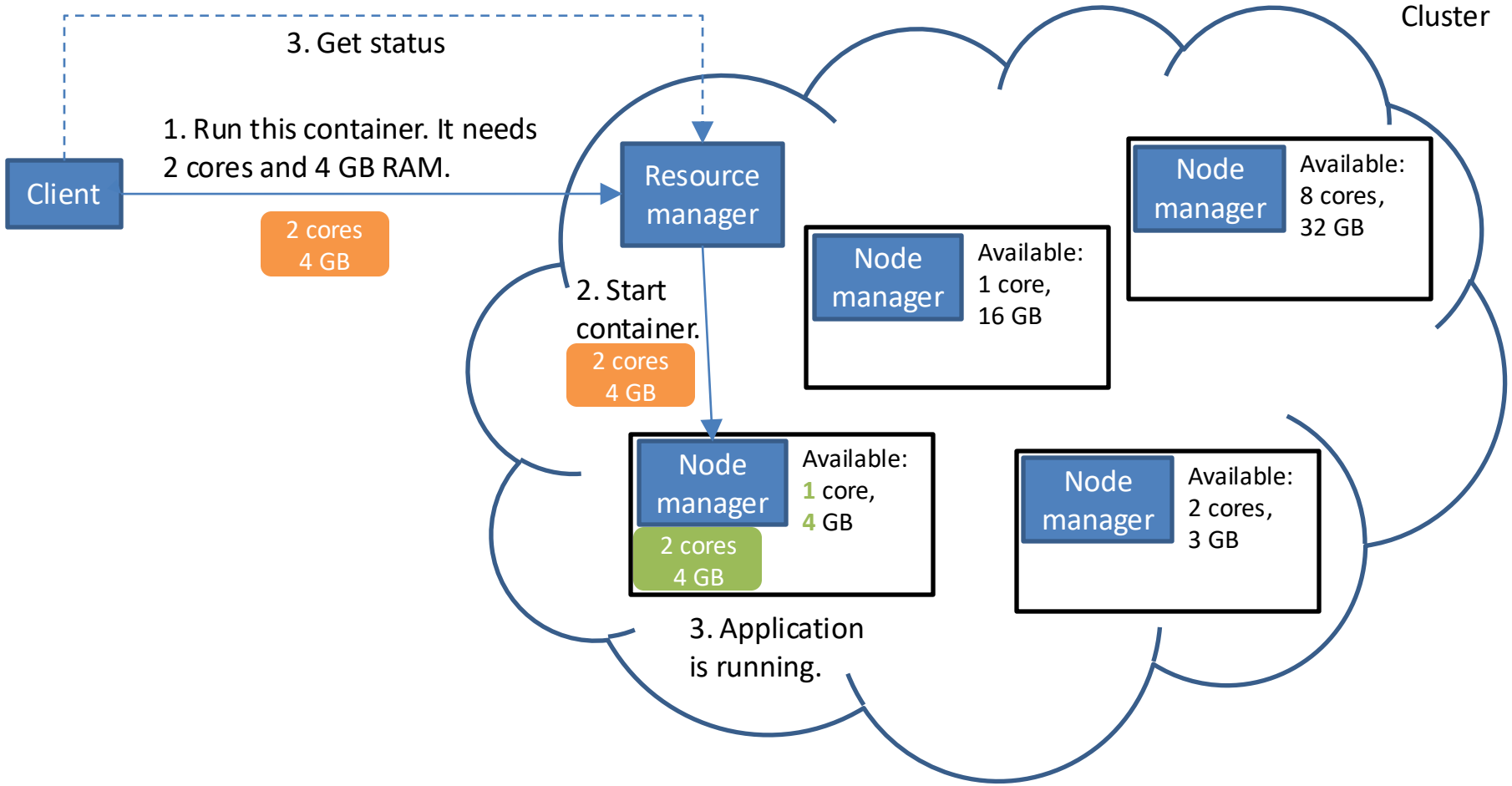
- Any machine with sufficient resources could run the application container. In practice, the scheduler will typically use a heuristic to make a choice such as:
  - For even load distribution, select the least loaded machine or the machine with the most resources available.
  - To avoid small resource leftovers, assign the application to the machine with the “best fit.” This means that the machine will have the least leftover resources.

- For our example, let’s assume the latter strategy is used. The resource manager will communicate with the corresponding node manager to start the container.

Basic View of Cluster Resource Management (cont.)



Basic View of Cluster Resource Management (cont.)



Scheduling Policies

- The resource manager also decides what to do when there are *insufficient* resources. It collects requests in a queue, where they wait until resources become available from completed or terminated jobs.

Scheduling Policies

- A scheduling policy decides which request will be next in line. Here are some examples:
  - First-in first-out (FIFO) is the simplest approach: requests are served in order of arrival.
  - FAIR: the next request served is for the user currently has the smallest share of cluster resources relative to what they're entitled to.
    - The goal is equalizing resource allocation over time, not just queue ordering.
  - Priority: requests with higher priority are served first.

How Many Resources to Request?

- The resource manager, together with the node managers, enforces resource consumption limits. This means that if an application exceeds its requested resources, it will be terminated.
  - However, asking for too much makes it harder to find a machine with enough memory and CPUs.
- Hadoop MapReduce and Spark can automatically determine the number of cores needed for application master and worker processes.

How Many Resources to Request?

- The user only needs to worry about the amount of memory.
  - A good programmer will analyze memory consumption of their program and then choose accordingly.

- In the worst case, one can apply trial-and-error: start with a "good guess," then increase container memory size if necessary.
  - If your Spark task processes a 2 GB partition and needs to hold intermediate data structures, requesting 4 GB gives reasonable headroom. If it gets killed, bump to 6 GB and resubmit.

Application Management

- A distributed data-processing job consists of many tasks that are running on different machines. These tasks must be coordinated, as we will discuss in a future module. At a quick glance:
  - Job = Entire process (i.e., given a text corpus, return how many times each word appears)
  - Task = Something that helps complete a job (i.e., given a line of text, split it to get each word independently)

Application Management

- Should the cluster resource manager handle managing jobs and tasks?

Application Management

- Should the cluster resource manager handle managing jobs and tasks?
  - Pro: We already have a centralized process that is aware of available resource, so let's use it. Old Hadoop versions took this route.
  - Con: The resource manager would have to be aware of application semantics. For MapReduce, it would have to understand in what order Map or Reduce tasks can be scheduled and what to do when one fails.

Application Management

- The current trend is to separate resource management from application management.

- This way a single generic resource manager like YARN can support diverse applications and services on the same cluster.

- Today a single YARN cluster might simultaneously run Spark batch jobs, Flink stream processing, etc., all without any of them needing to know about each other.

Spark Application Management

- To see how resource and application management interact, we take a closer look at the execution of a Spark job.

- You will become more familiar with terminology like job, task, and executor as we progress in the course.

Key Terms

- A Spark job runs inside a YARN cluster using four key pieces:
  - **Container:** A reserved slice of a node's CPU and memory. Everything runs inside a container.
  - **Application Master (AM):** Negotiates with the resource manager to acquire and manage containers for the job. It handles resource requests and container failures.
  - **Driver:** Plans and coordinates the actual Spark computation. It builds the execution plan, assigns tasks to executors, and collects results.
  - **Executor:** A worker process that lives in its own container. It receives tasks from the driver, runs them, and reports results back.

Spark Application Management

- The client submits the job to the resource manager. After that, communication splits into two layers:
  - **Resource layer:** The AM requests containers from the resource manager. The resource manager assigns them. That's it — the resource manager is done.
  - **Application layer:** The driver (inside the AM) sends tasks to executors and collects results. The resource manager is not involved in any of this.
- This is the clean separation: the resource manager allocates boxes, the application master decides what happens inside them.

Spark Application Management

- There is exactly one Application Master per Spark job. This follows the same pattern as GFS and HDFS — a single coordinator avoids the complexity of distributed consensus.

- If 10 Spark jobs are running on the cluster, there are 10 Application Masters, each managing its own job independently.

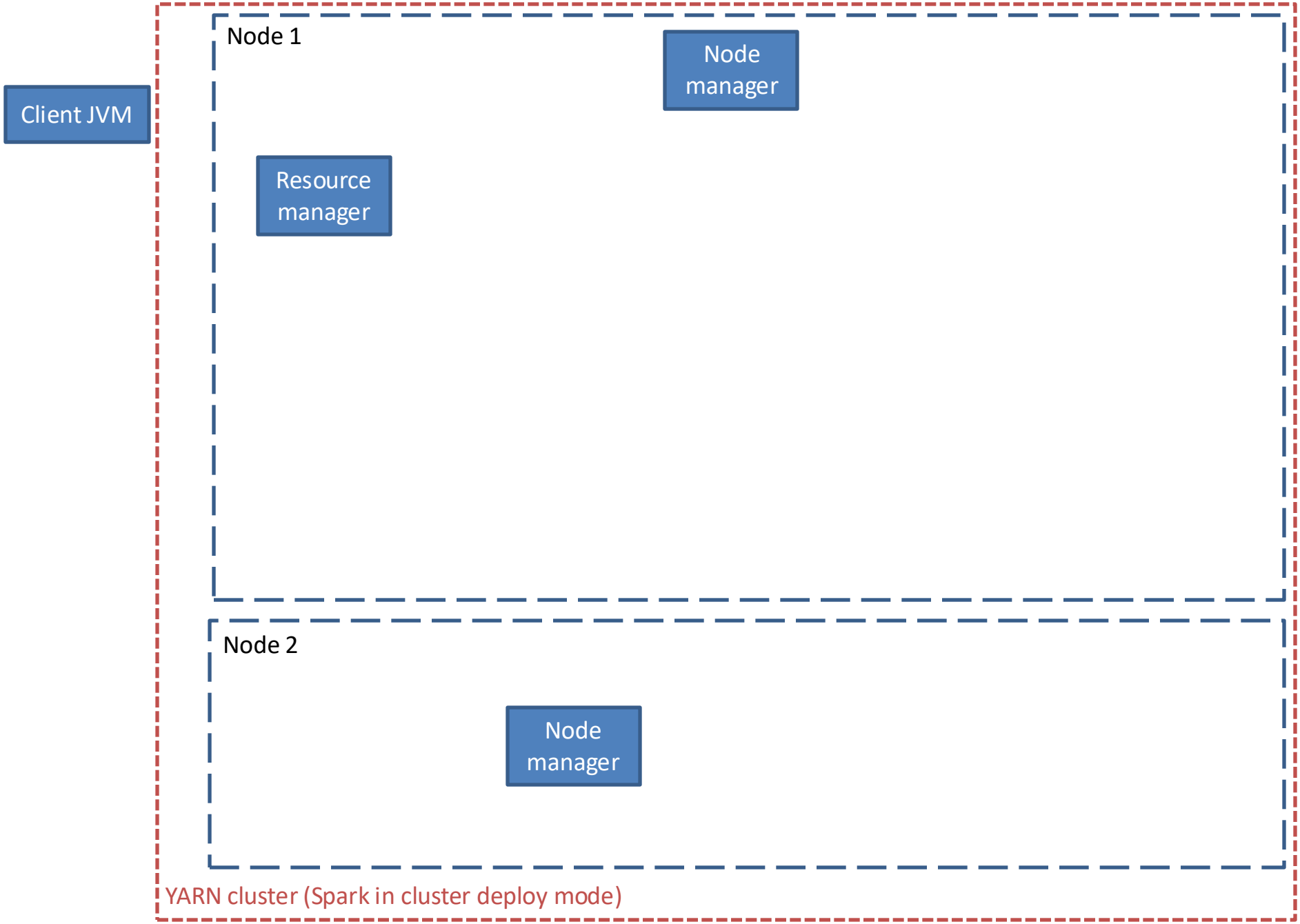


Illustration based on Zecevic/Bonaci book

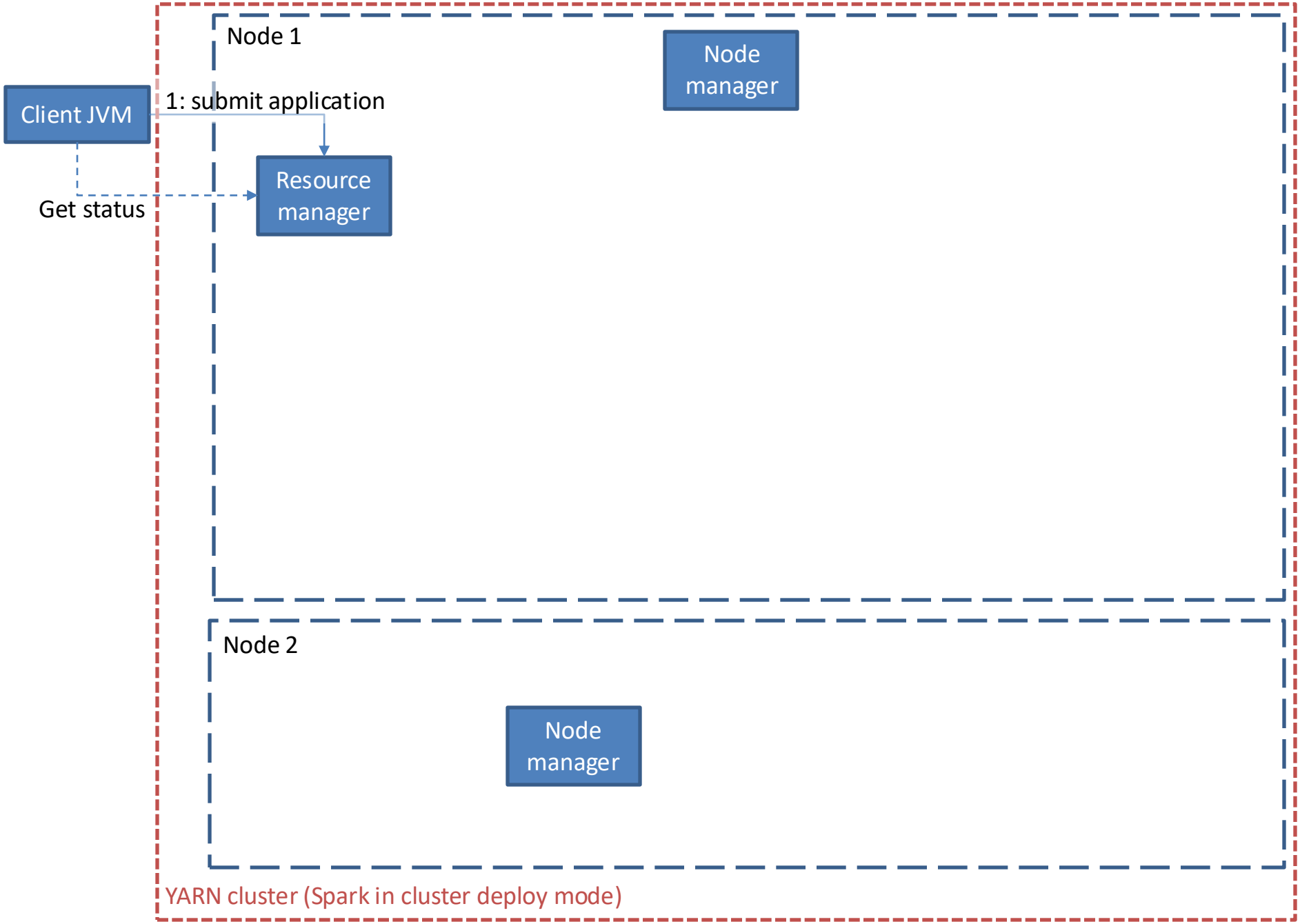


Illustration based on Zecevic/Bonaci book

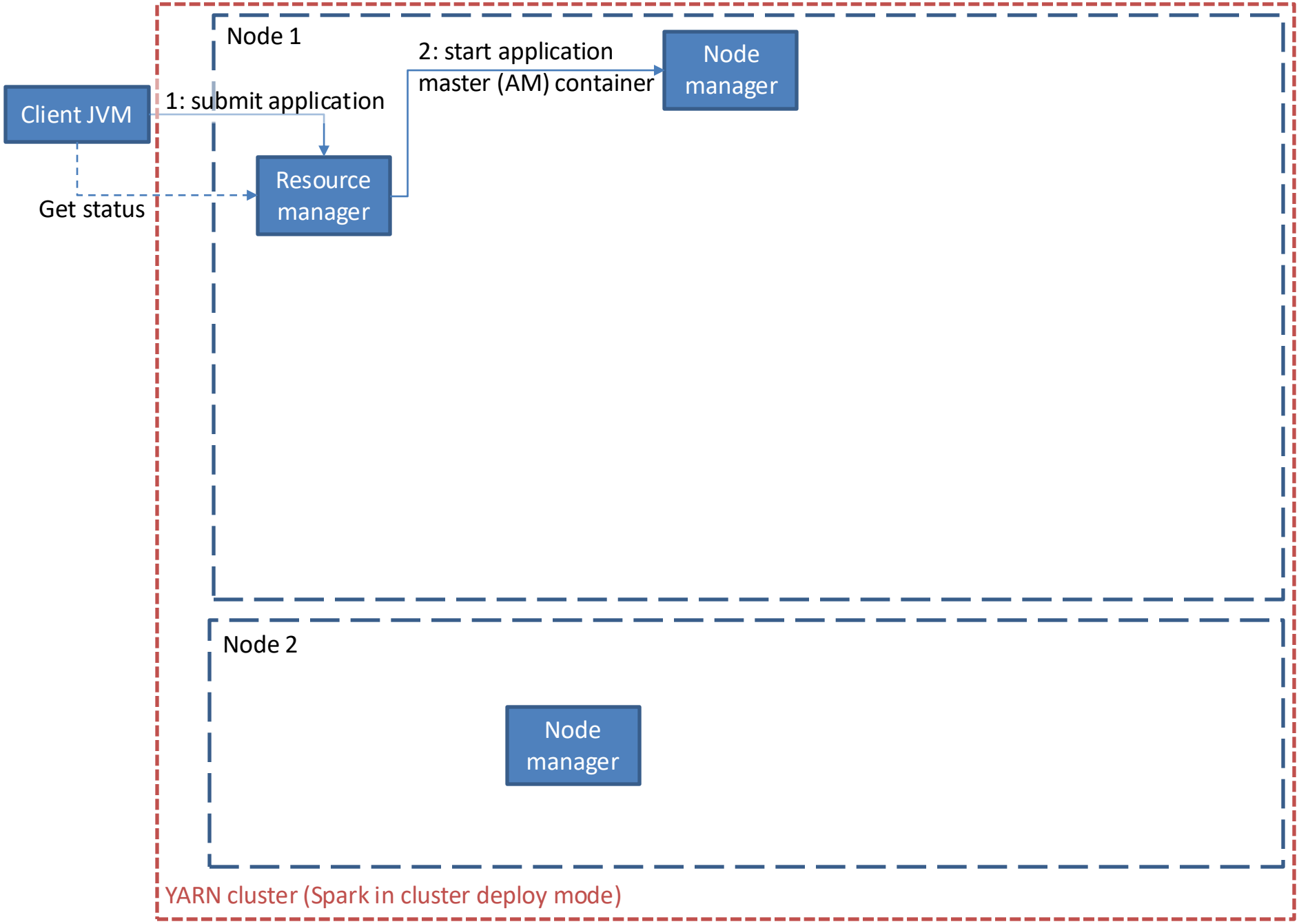


Illustration based on Zecevic/Bonaci book

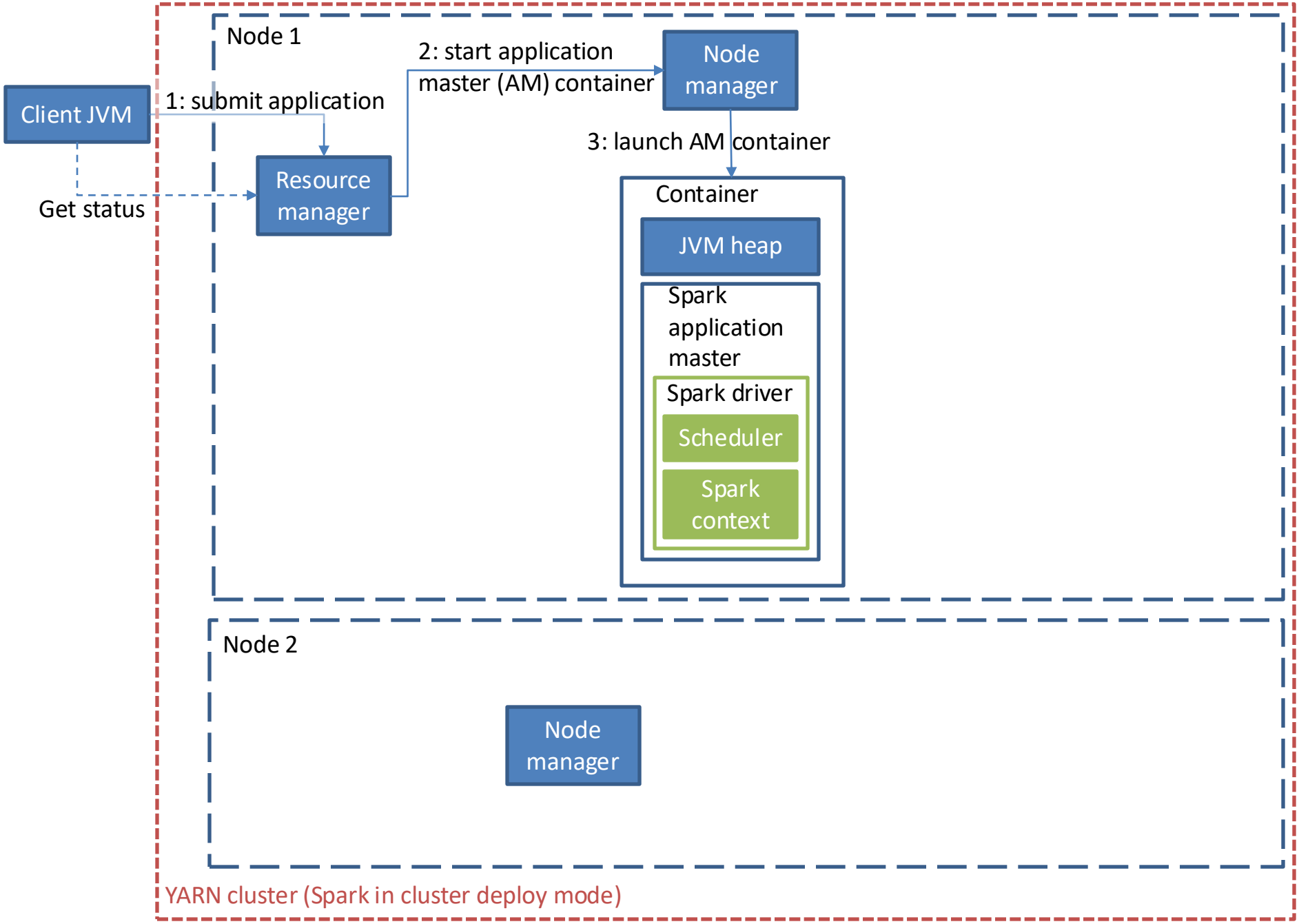


Illustration based on Zecevic/Bonaci book

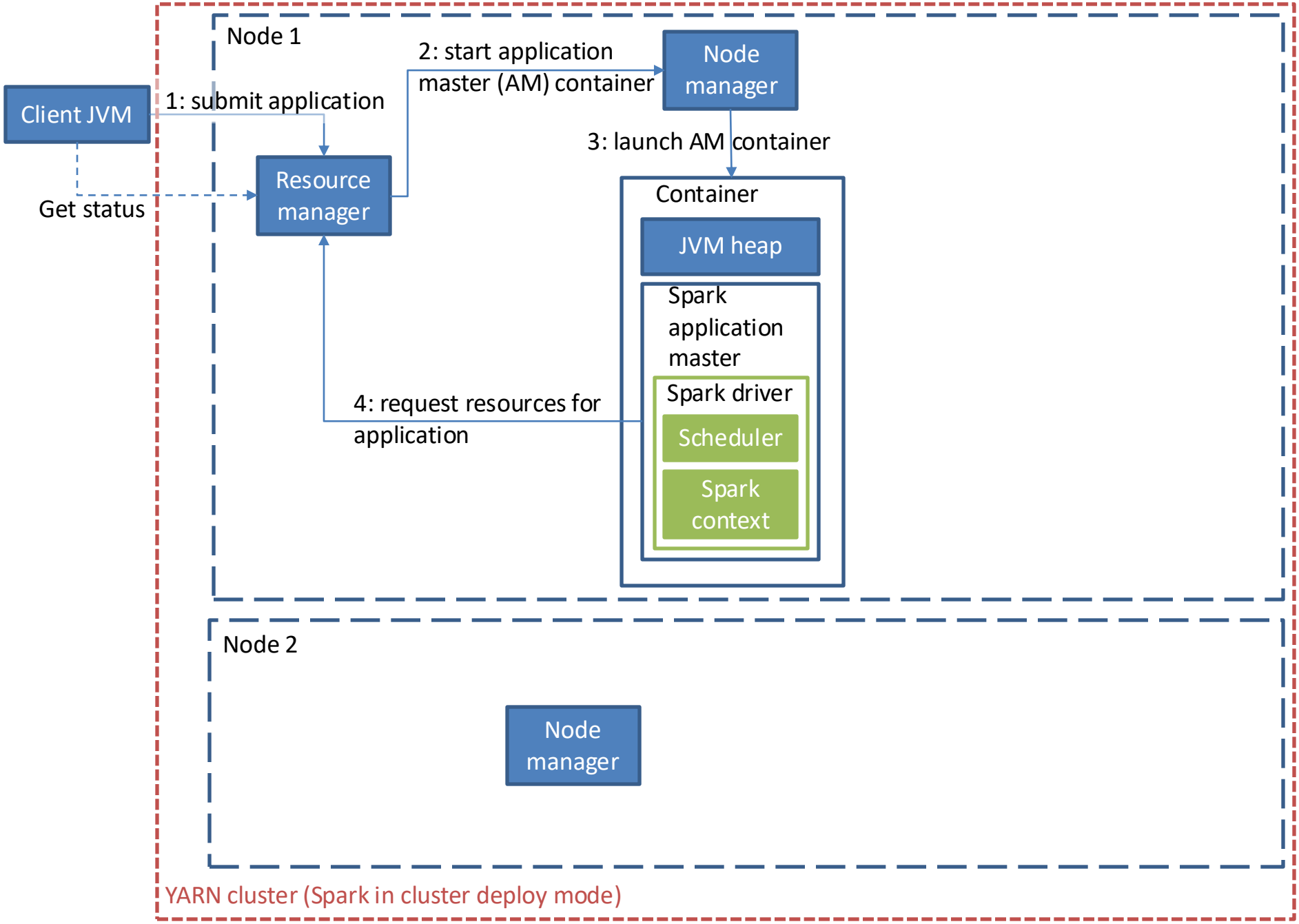


Illustration based on Zecevic/Bonaci book

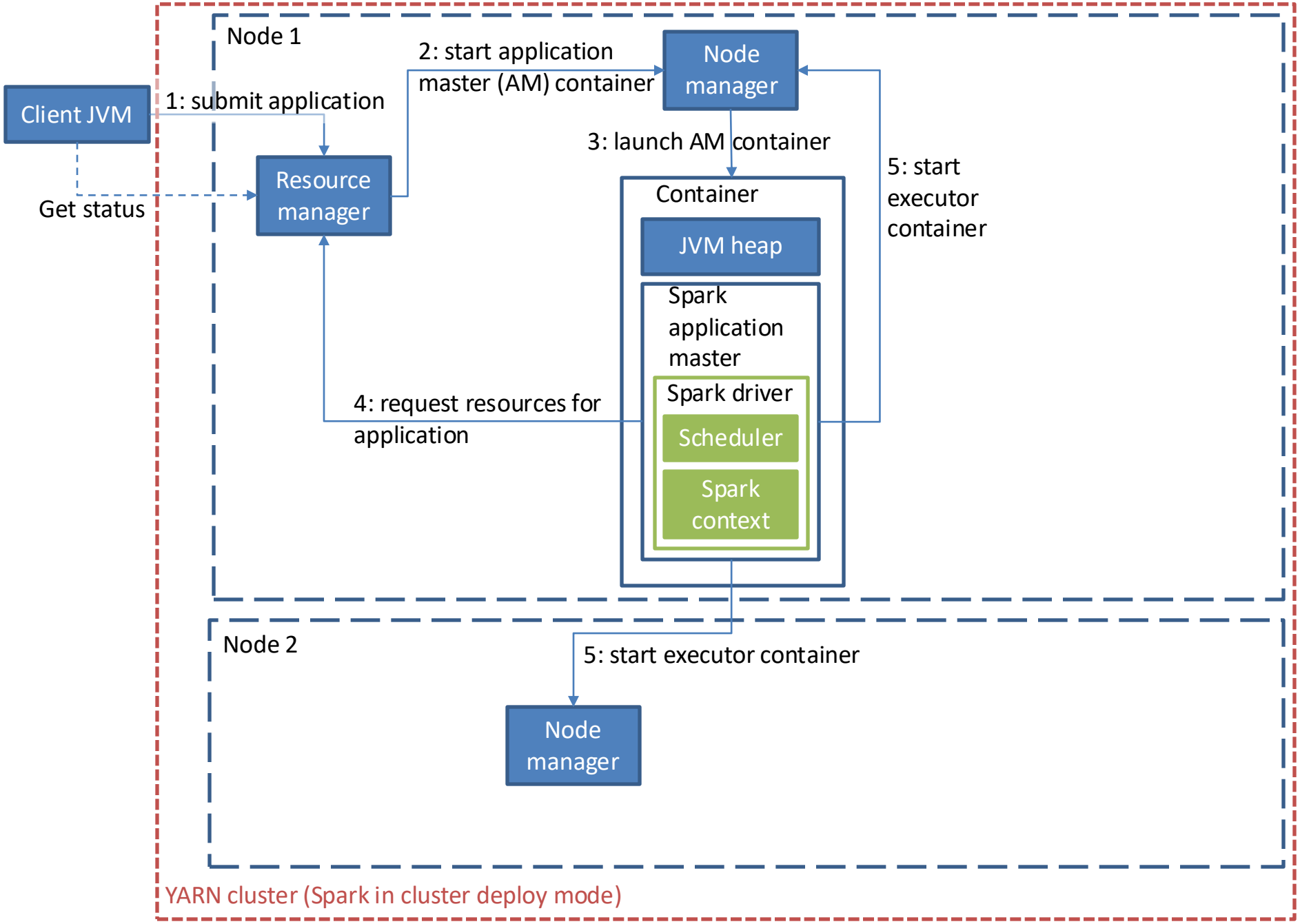


Illustration based on Zecevic/Bonaci book

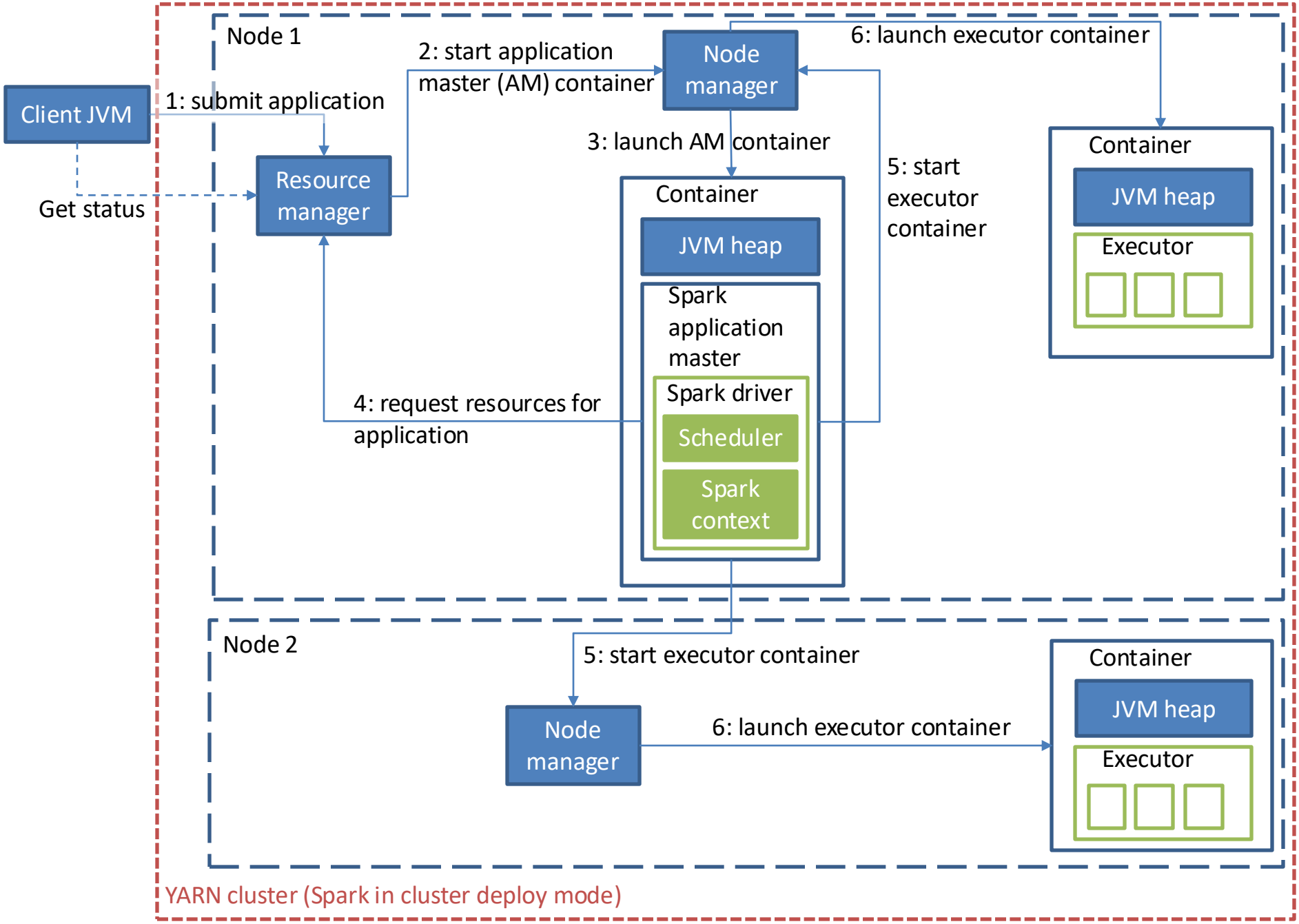


Illustration based on Zecevic/Bonaci book

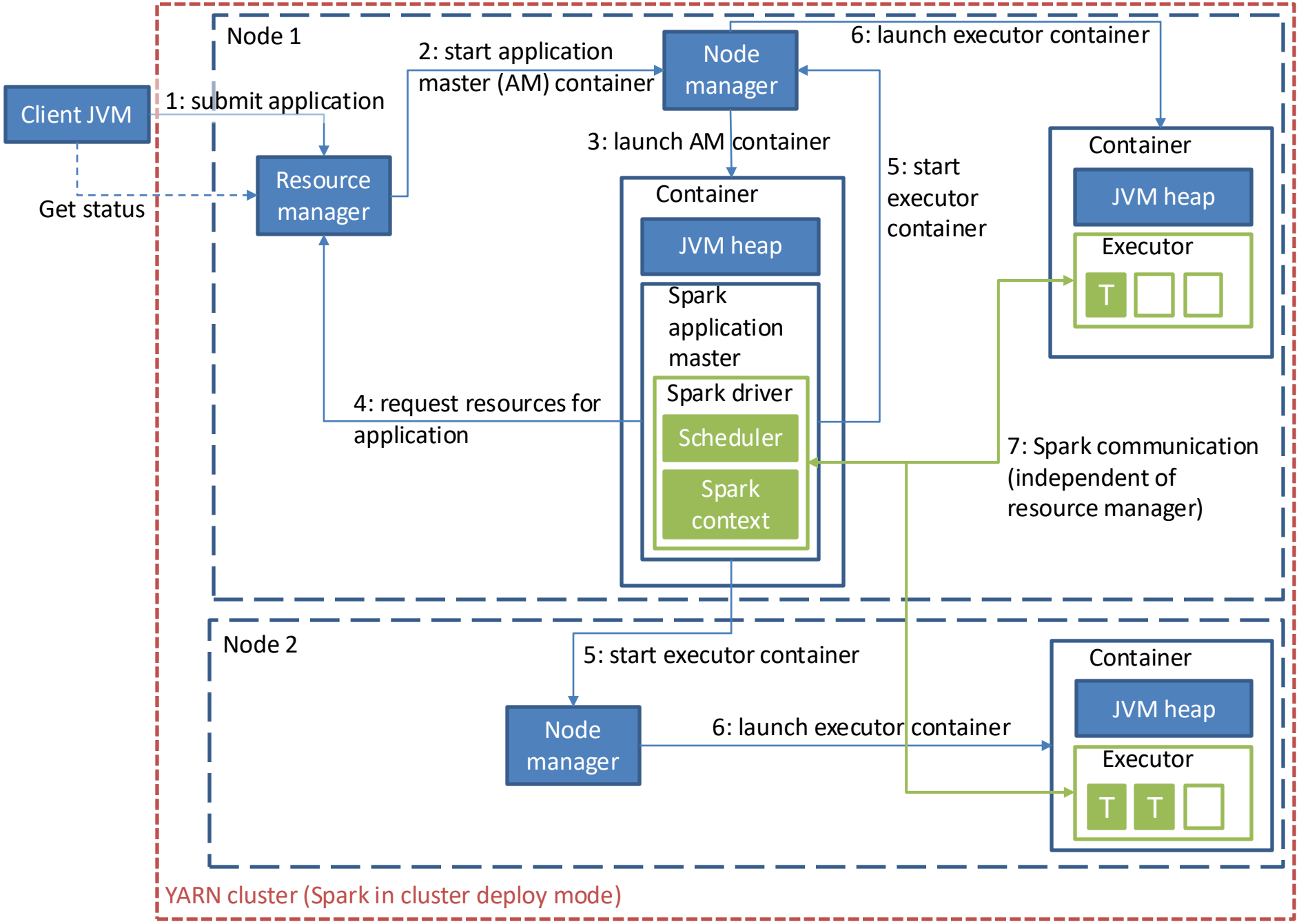


Illustration based on Zecevic/Bonaci book

References

- Petar Zecevic and Marko Bonaci: Spark in Action. Manning Publications, 2016