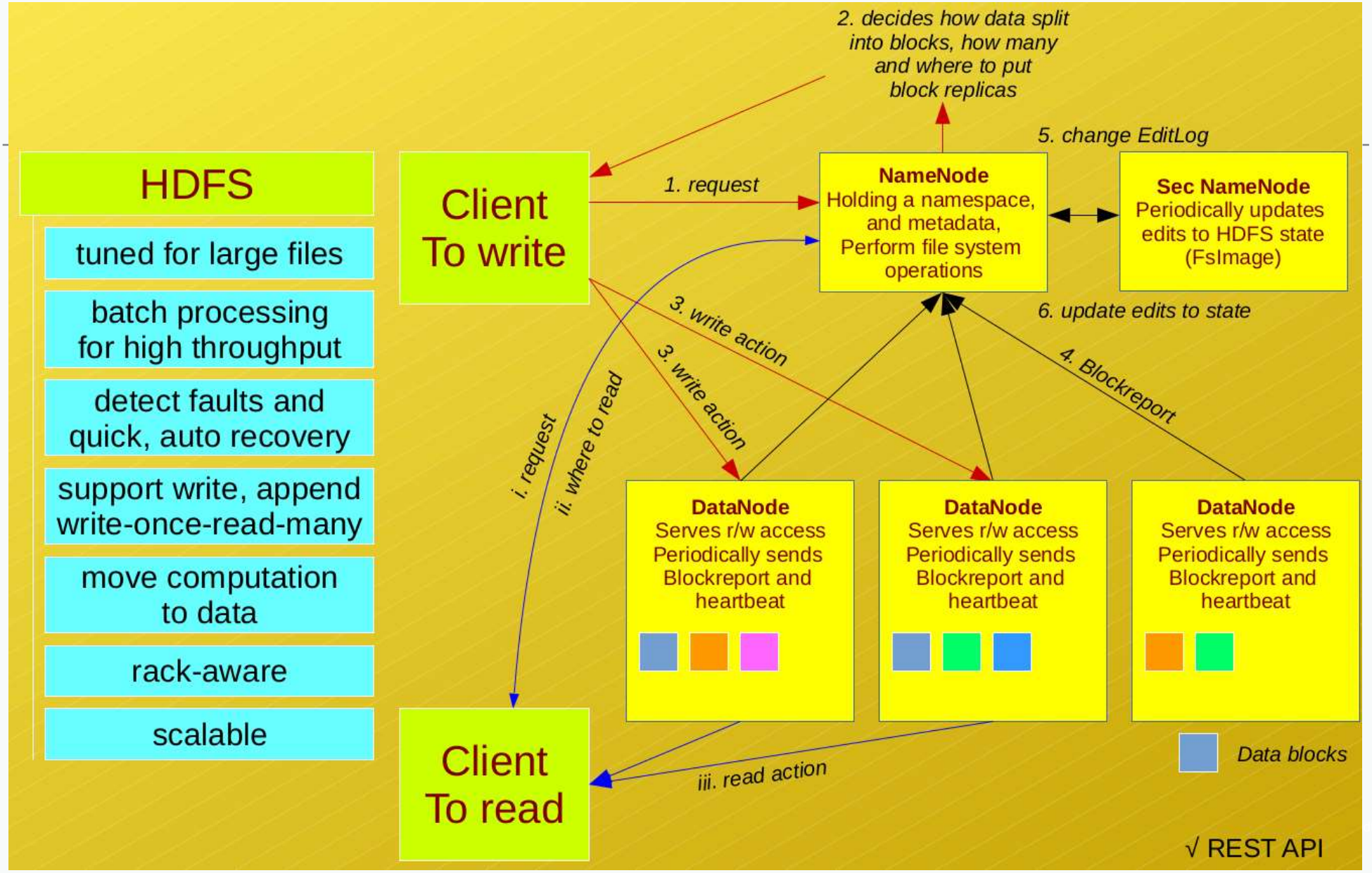
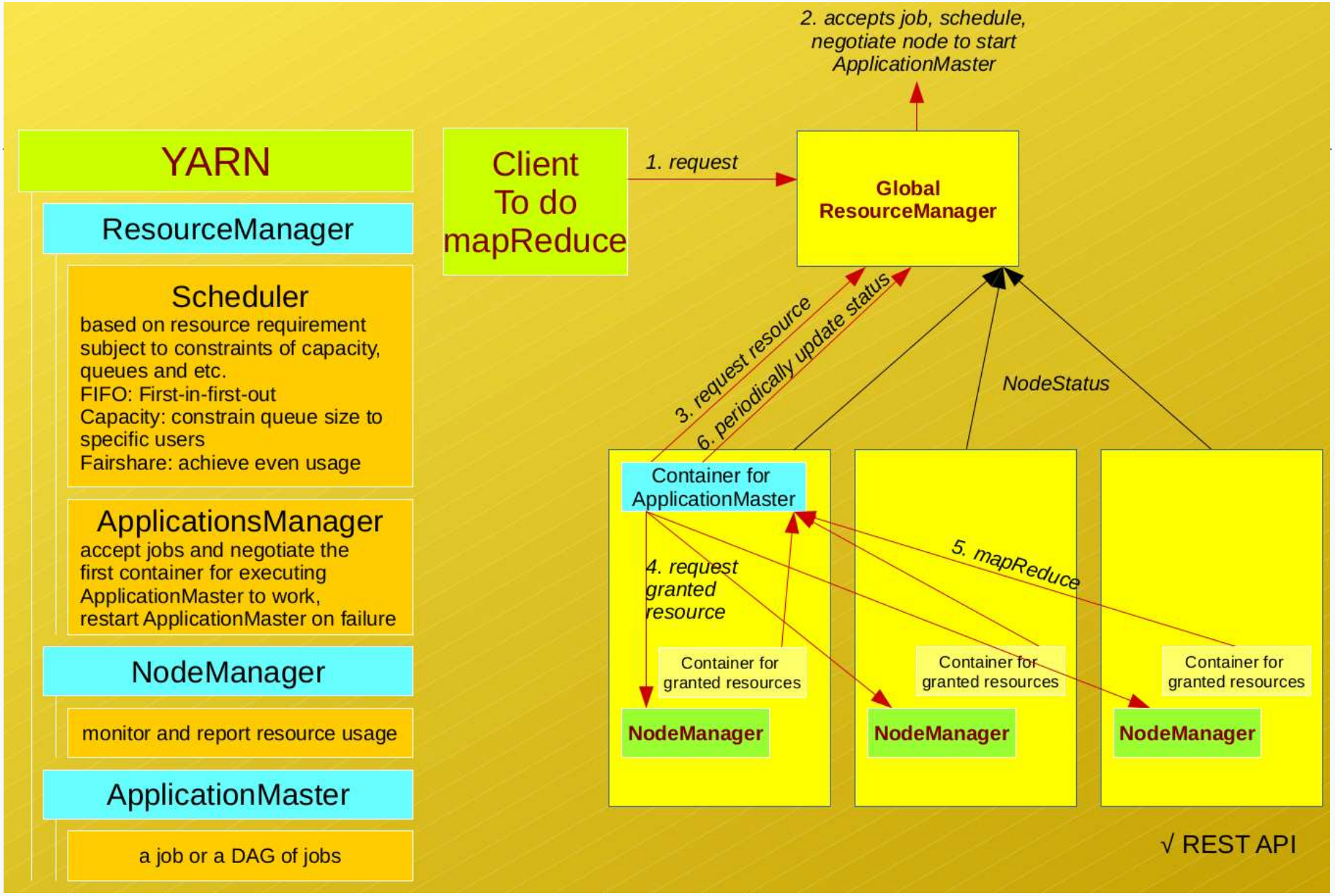
YARN

YARN Introduction

- Cluster resource management layer of Hadoop
- Does:
  - Resource management
  - Job Scheduling
- YARN has these services:
  - Global Resource Manager
    - To distribute resources among various applications. It has below functionalities
    - Scheduler
    - Application Manager
  - Node Manager
    - Per machine slave daemon
  - Application Master
    - Per application specific entity
    - Negotiate the resources for execution from Resource Manager
    - Works with Node Manager for executing and monitoring component tasks.


HDFS
tuned for large files
batch processing for high throughput
detect faults and quick, auto recovery
support write, append write-once-read-many
move computation to data
rack-aware
scalable
Client To write
Client To read
2. decides how data split into blocks, how many and where to put block replicas
1. request
NameNode
Holding a namespace, and metadata, Perform file system operations
Sec NameNode
Periodically updates edits to HDFS state (FsImage)
5. change EditLog
6. update edits to state
3. write action
3. write action
4. Blockreport
i. request
ii. where to read
DataNode
Serves r/w access
Periodically sends Blockreport and heartbeat
DataNode
Serves r/w access
Periodically sends Blockreport and heartbeat
DataNode
Serves r/w access
Periodically sends Blockreport and heartbeat
iii. read action
Data blocks
√ REST API


YARN
ResourceManager
Scheduler
based on resource requirement subject to constraints of capacity, queues and etc.
FIFO: First-in-first-out
Capacity: constrain queue size to specific users
Fairshare: achieve even usage
ApplicationsManager
accept jobs and negotiate the first container for executing ApplicationMaster to work, restart ApplicationMaster on failure
NodeManager
monitor and report resource usage
ApplicationMaster
a job or a DAG of jobs
Client To do mapReduce
1. request
2. accepts job, schedule, negotiate node to start ApplicationMaster
Global ResourceManager
3. request resource
6. periodically update status
NodeStatus
Container for ApplicationMaster
4. request granted resource
5. mapReduce
Container for granted resources
Container for granted resources
Container for granted resources
NodeManager
NodeManager
NodeManager
√ REST API

Sqoop
Data transferred between Hadoop and structured DB

Flume
Stream web logs and events, aggregate, store in Hadoop

HDFS
Distributed file system

YARN
Resource Management, Job scheduling and monitoring

Spark (atop HDFS, or YARN)
General engine for data analysis, ML, ETL, stream processing, graph analysis...

Impala
Analytic SQL query engine

Giraph
Iterative graph processing system

Tez (optional)
DAG execution engine

MapReduce
Distributed processing

Mahout (Scala)
Machine Learning platform

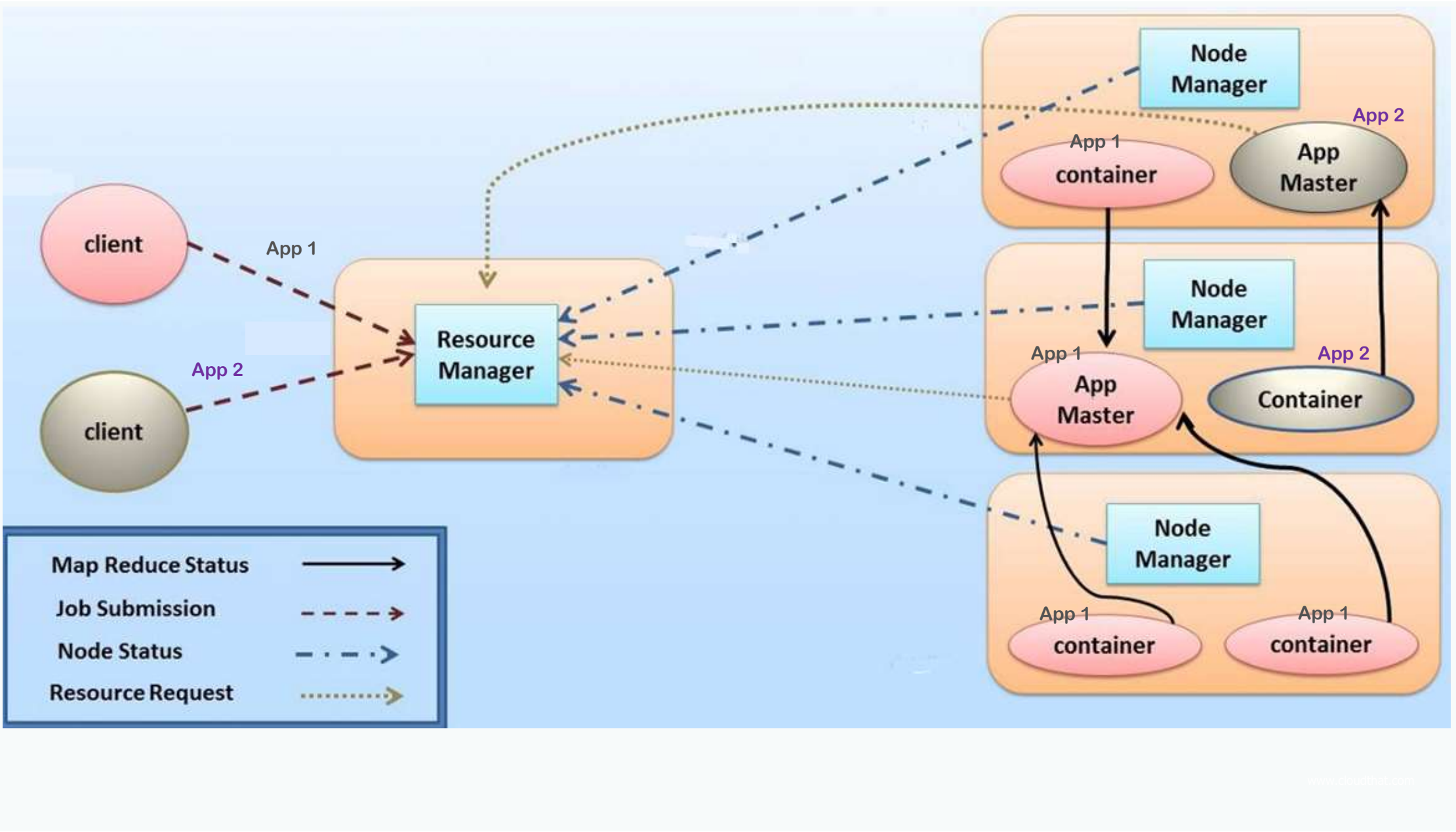
Pig (Pig Latin)
ETL, Data analysis platform

Oozie
workflow and scheduler

Hive
Data warehouse on SQL

HBase
Distributed non-relational database for real-time access to big data

ZooKeeper
Cluster and config management, Coordination, locking, and synchronization service


client
App 1
client
App 2
Resource Manager
Node Manager
App 2
App 1
container
App Master
Node Manager
App 1
App Master
App 2
Container
Node Manager
App 1
container
App 1
container
Map Reduce Status
Job Submission
Node Status
Resource Request

Global Resource Manager

- To distribute resources among various applications in the system. It has two main functionalities.
- Scheduler:
  - Decides allocation of resources to various running applications.
- Application Manager:
  - Accepts the job submissions
  - Negotiating resources for executing the Application Master
  - Restarting the Application Master in case of failure

Node Manager

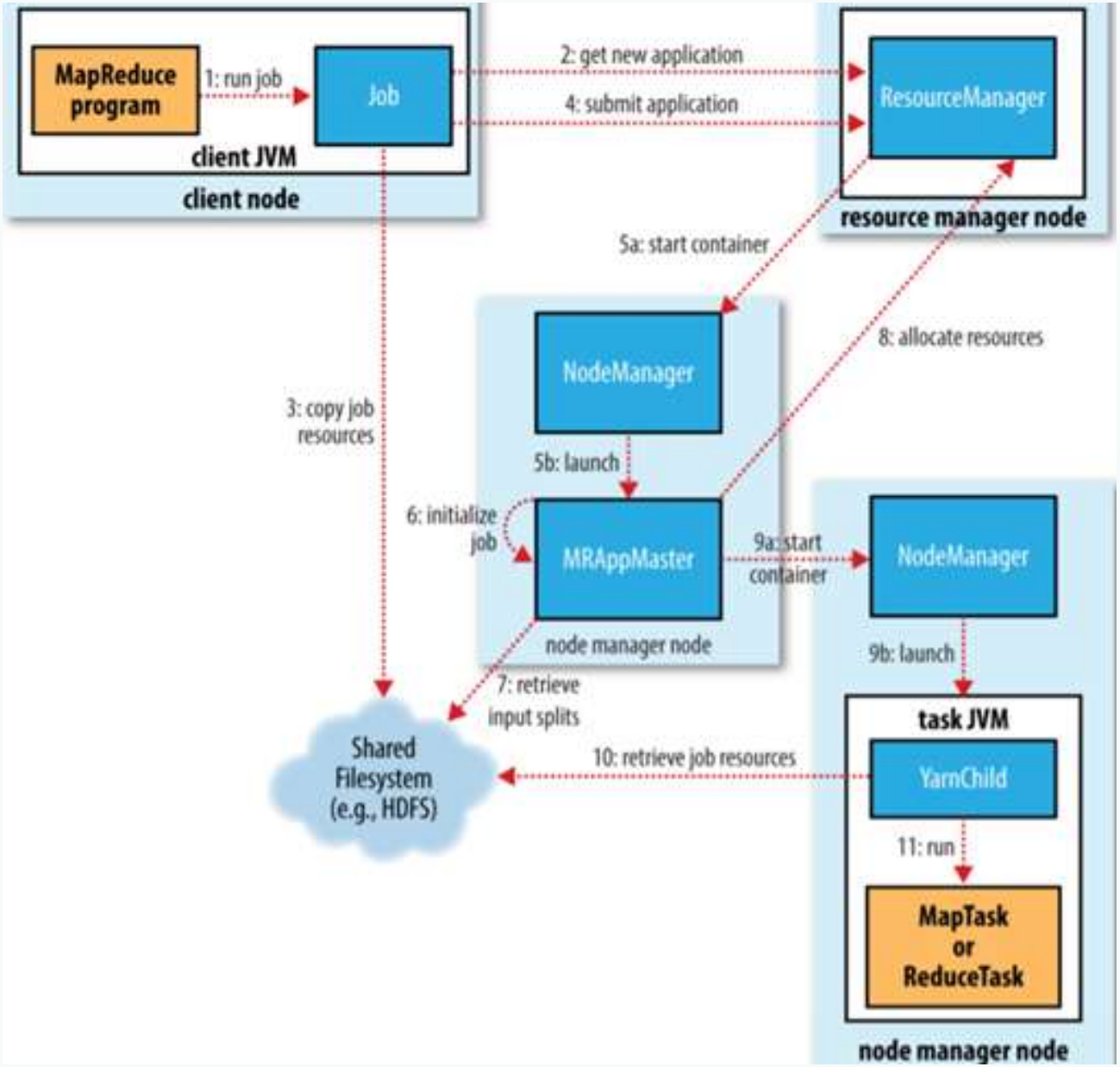
- Per machine slave daemon
- Responsible for launching the application containers for app execution
- Monitors the resource usage such as memory, CPU, disk etc.
- Reports the usage of resources to global Resource Manager.
- All nodes of the cluster have a certain number of containers.
- Container
  - Computing units
  - A kind of wrappers for node resources to perform tasks of a user application.
  - Responsible for running an application.
  - Apps run inside them
  - Ensures security and prevent apps from using resources or memory which aren't allocated to them
  - Apps are allowed to use limited & fixed amount of memory shared to them.

Application Master

- Per application specific entity
- Responsible for negotiating the resources for execution from Resource Manager
- Works with Node Manager for executing and monitoring component tasks.
- Each app running inside a container (or in multiple containers) has one application master
- It keeps track of app's progress.
- It registers itself with the resource manager (whenever new app is being run) and communicates with it, giving information about app's progress
- The manager , in turn, services requests raised by the client regarding the app.

YARN architecture for running a Map Reduce Job

- Five independent entities
- The client
  - Submits the MapReduce job
- The YARN resource manager
  - Coordinates the allocation of compute resources on the cluster
- The YARN node managers
  - Launch and monitor the compute containers on machines in the cluster.
- The MapReduce application master
  - Coordinates the tasks running the Map Reduce job
  - The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The distributed filesystem, which is used for sharing job files between the other entities



- Job Submission – 1, 2, 3, 4
- Job Initialization – 5, 6, 7
- Task Assignment – 8
- Task Execution – 9a, 9b, 10, 11
- Job Completion

Job Submission

- Asks the resource manager for a new application ID (step 2)
- Checks the output specification of the job. For example, if the output directory has not been specified or it already exists, the job is not submitted, and an error is thrown to the MapReduce program.
- Computes the input splits for the job. If the splits cannot be computed (because the input paths don’t exist, for example), the job is not submitted, and an error is thrown to the MapReduce program.
- Copies the resources needed to run the job, including the job JAR file, the configuration file, and the computed input splits, to the shared filesystem in a directory named after the job ID (step 3)
- Submits the job on the resource manager (step 4)

Job Initialization

- When the resource manager receives a call, it hands off the request to the YARN scheduler
- The scheduler allocates a container, and the resource manager then launches the application master's process.
- The application master will receive progress and completion reports from the tasks (step 6)
- Next, it retrieves the input splits computed in the client from the shared filesystem (step 7)
- It then creates a map task object for each split, as well as a number of reduce task
- Tasks are given IDs at this point.
- The application master must 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

Task Assignment

- If the job does not qualify for running in same JVM, then the application master requests containers for all the map and reduce tasks in the job from the resource manager (step 8).
- Requests for map tasks are made first and with a higher priority
- Requests for map tasks have data locality constraints that the scheduler tries to honor.
- In the optimal case, the task is data local that is, running on the same node that the split resides on. Alternatively, the task may be rack local: on the same rack, but not the same node, as the split. Some tasks are neither data local nor rack local and retrieve their data from a different rack than the one they are running on.
- Requests also specify memory requirements and CPUs for tasks.

Task Execution

- Once a task has been assigned resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager (steps 9a and 9b)
- The task is executed by a Java application
- Finally, it runs the map or reduce task (step 11)

Job Completion

- When the application master 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.
- Job statistics and counters are printed to the console at this point.
- Finally, on job completion, the application master and the task containers clean up their working state

Thank You