Master 2 Informatique

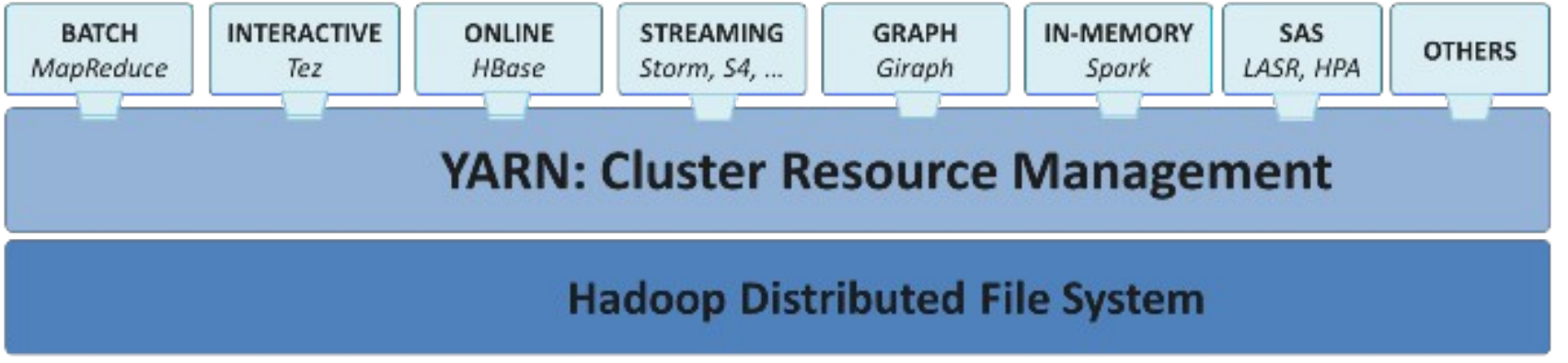
Frédéric Raimbault

Université de Bretagne Sud – Vannes

- *Apache Hadoop YARN*, A. Murphy and al., Addison Wesley
- *YARN Essentials*, A. Fasale and al., Packt Publishing

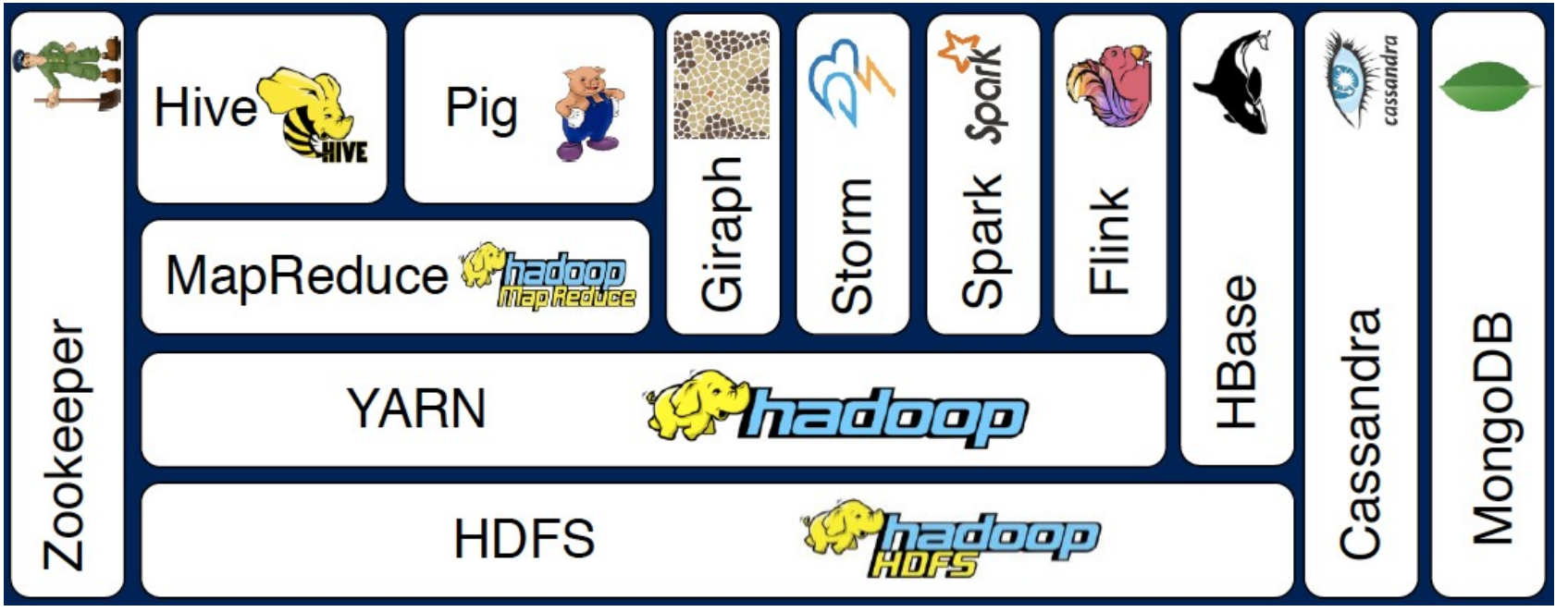
What is YARN ?

- A generic resource manager in a typical cluster
  - Conceived and architected at Yahoo! (2008)
  - Introduced with Hadoop 2.x, aka MRv2 (2013)
- Yarn allows different data processing engines to run and process data stored in HDFS.



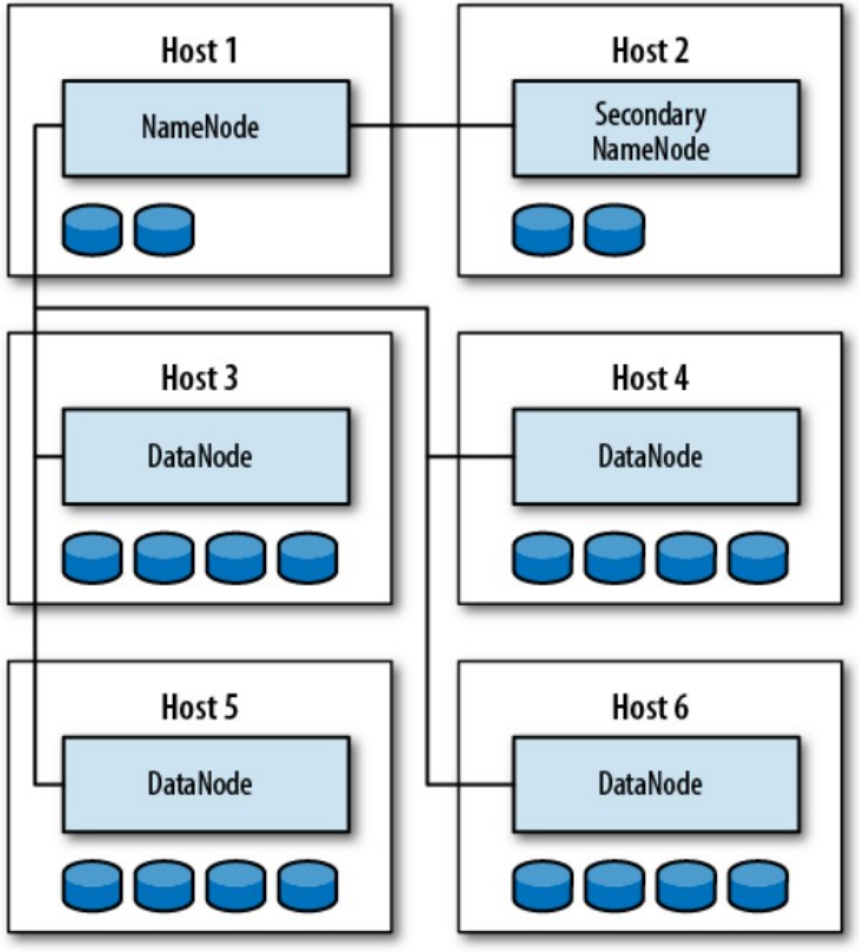
Hadoop World

- Apache Hadoop: an open-source platform for big data processing



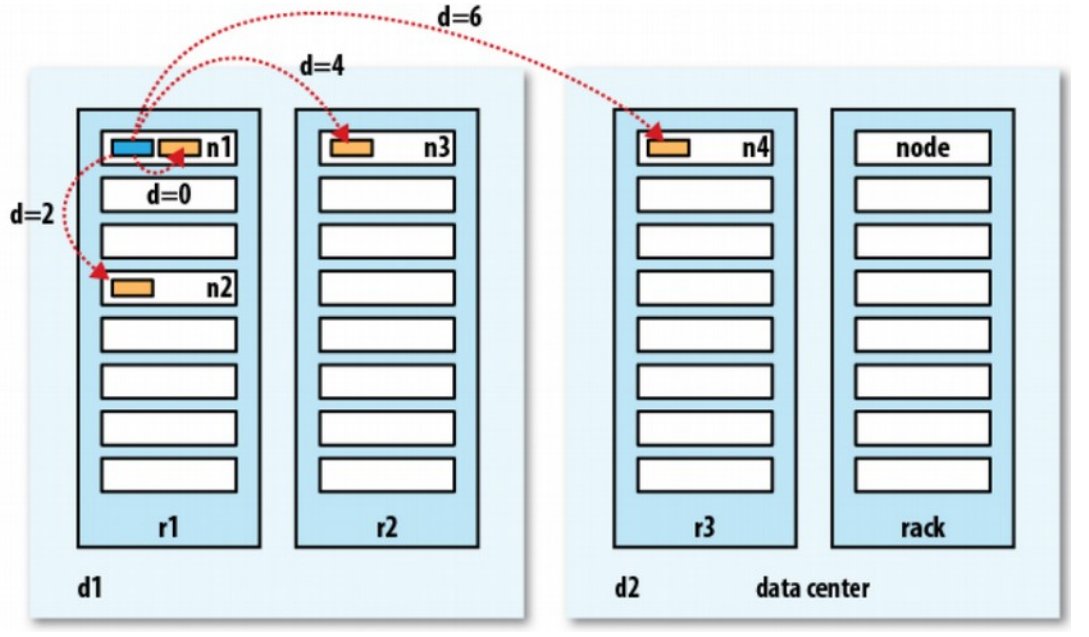
HDFS Architecture

- NameNode (master)
  - 1 per cluster
  - Stores metadata and map (file to block)
  - Secondary NameNode
    - 1 per cluster
    - Checkpoint
- DataNode (slaves)
  - 1 per node
  - Store block data



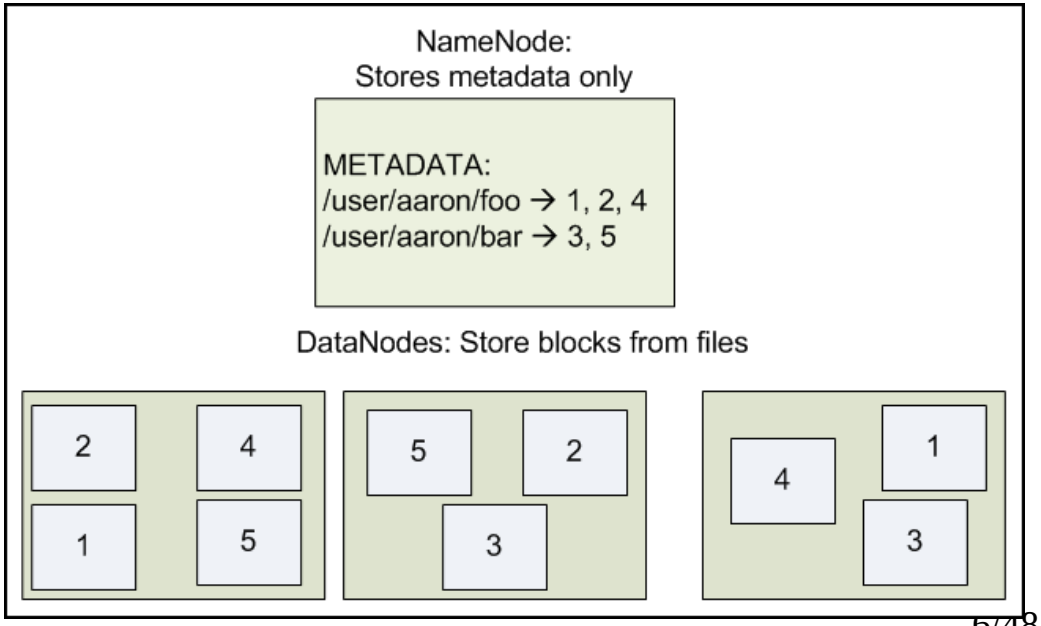
HDFS Blocks

- Data is split into fixed size blocks (64Mo, 128 Mo)
- Data is distributed and replicated
  - Typical block replication of 4
  - Distance is taken into account for performance



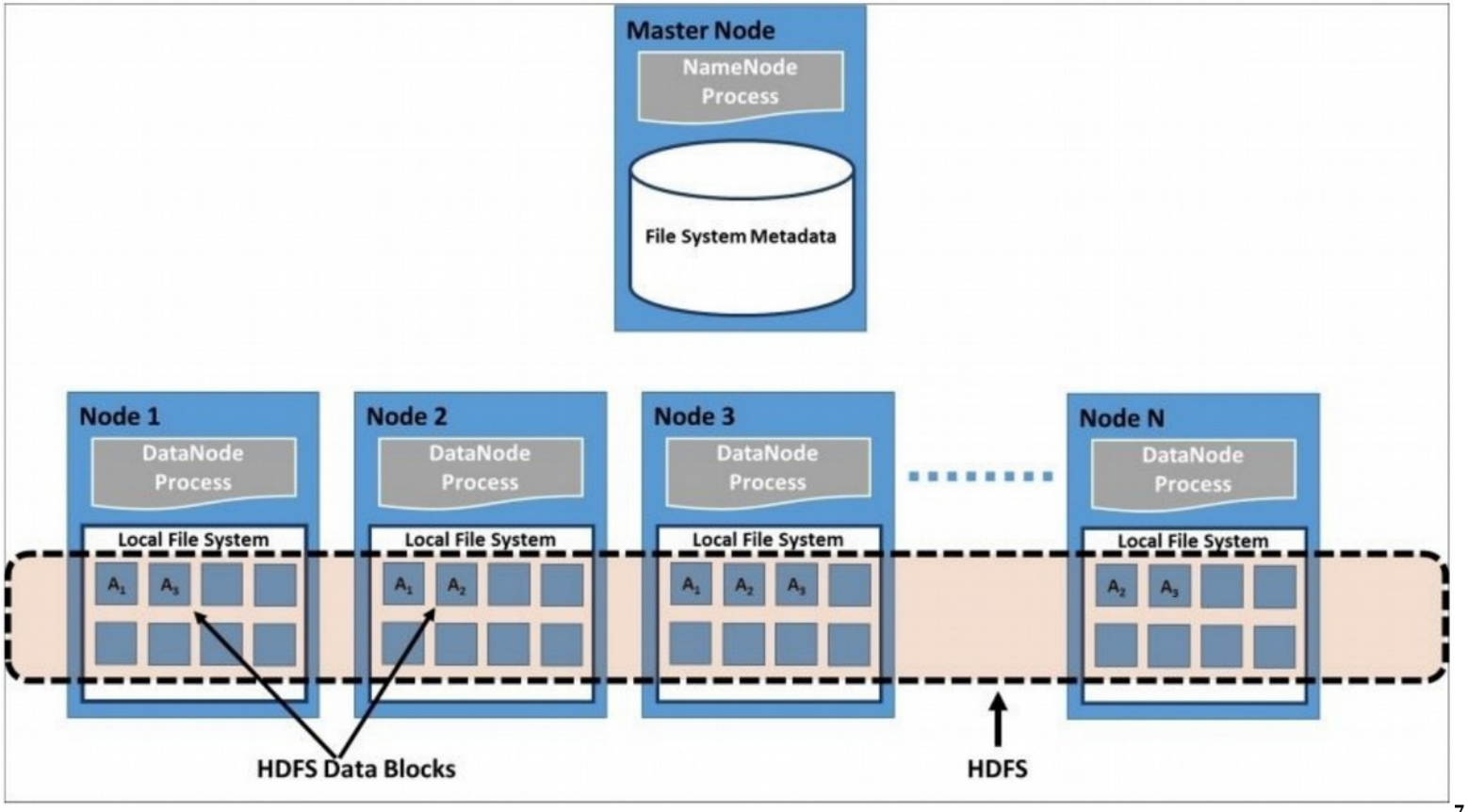
HDFS Meta Data

- Hierarchical file system
  - Pathname, filename, owner, groups, perms.
- Separate name space isolated from local FS
- Block map
  - Locations of blocks and locations of replicated blocks



HDFS Blocks Data

- User space distributed file system



HDFS Clients

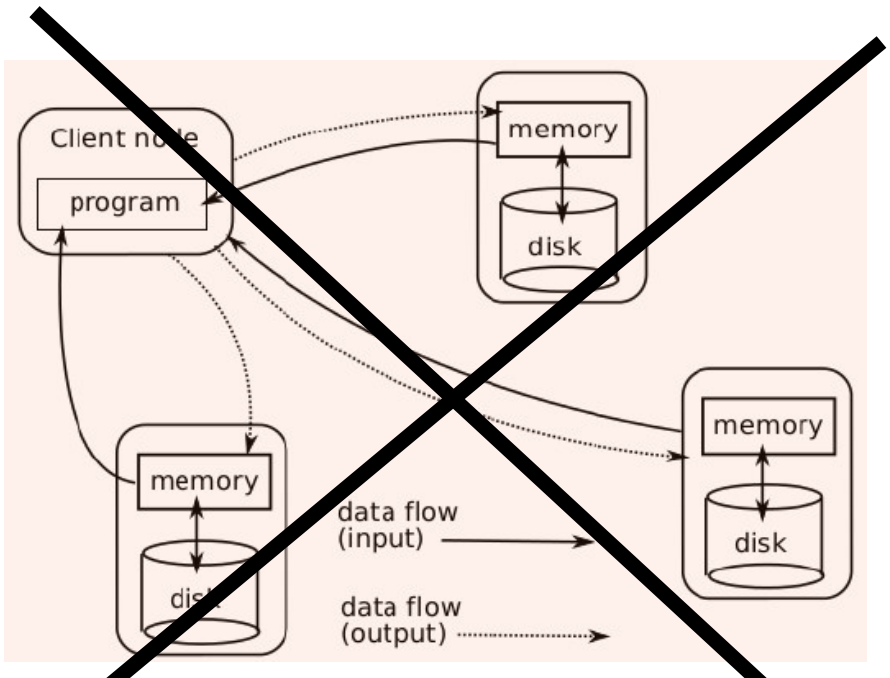
- Command-line tools

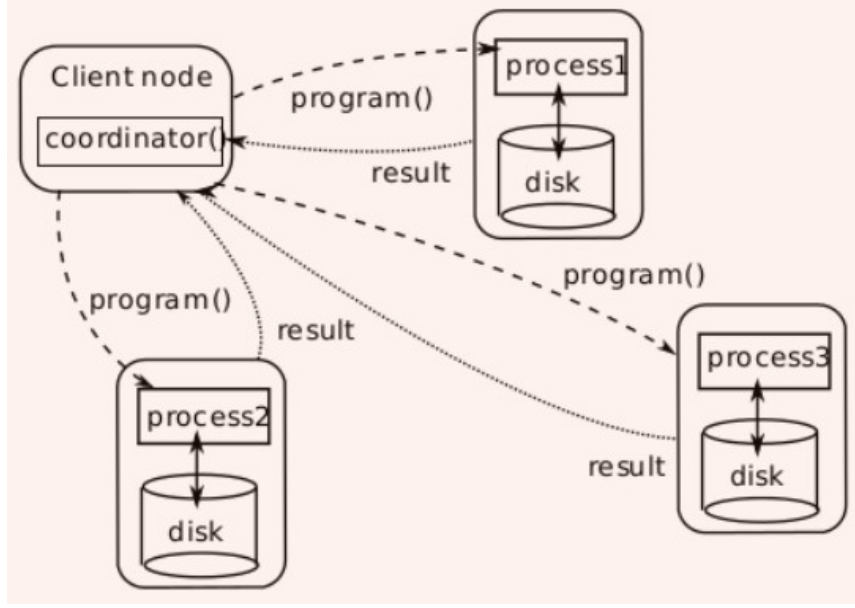
```
hadoop fs -help
```

- Java API
  - `org.apache.hadoop.fs.FileSystem`
    - `FSDataOuputStream create()`
    - `copyFromLocalFile(), copyToLocalFile()`
    - `getFileBlockLocations()`
    - `...`
- HTTP server
  - http://cluster-pedago.univ-ubs.fr/wiki/doku.php/hdfs

Big Data (Batch) Processing

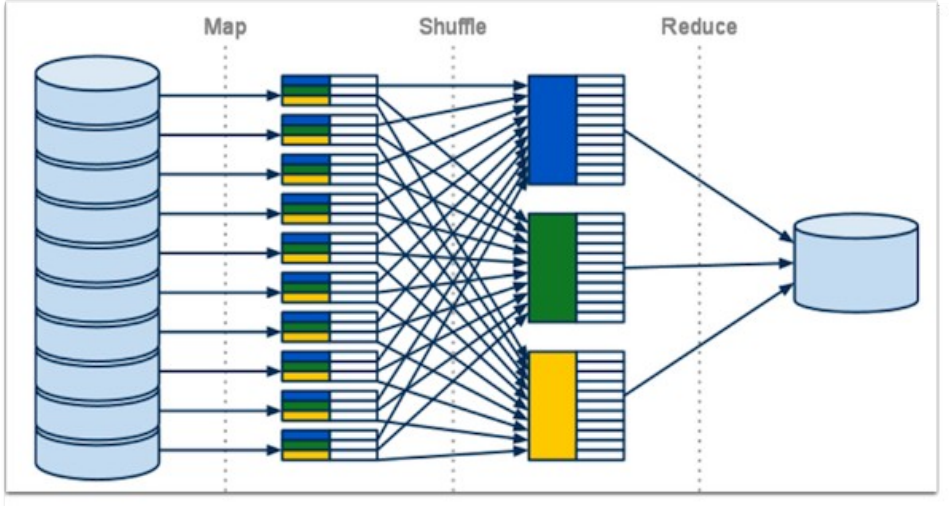
- Partition data across nodes
- ~~Remote data access and locally processing~~
- Move applications to the data and process data remotely

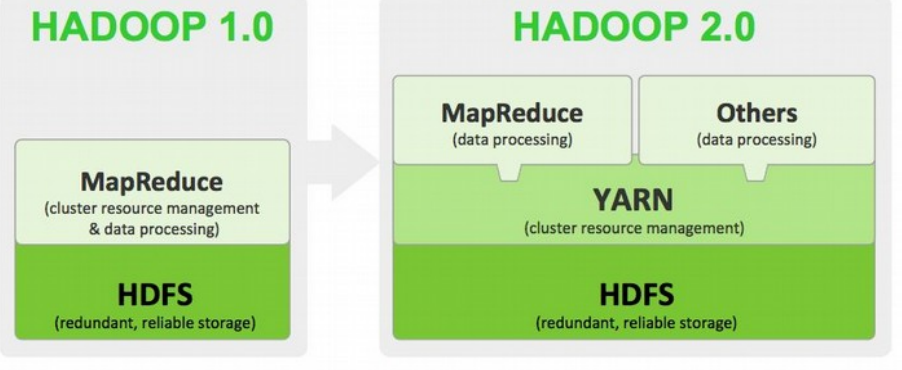




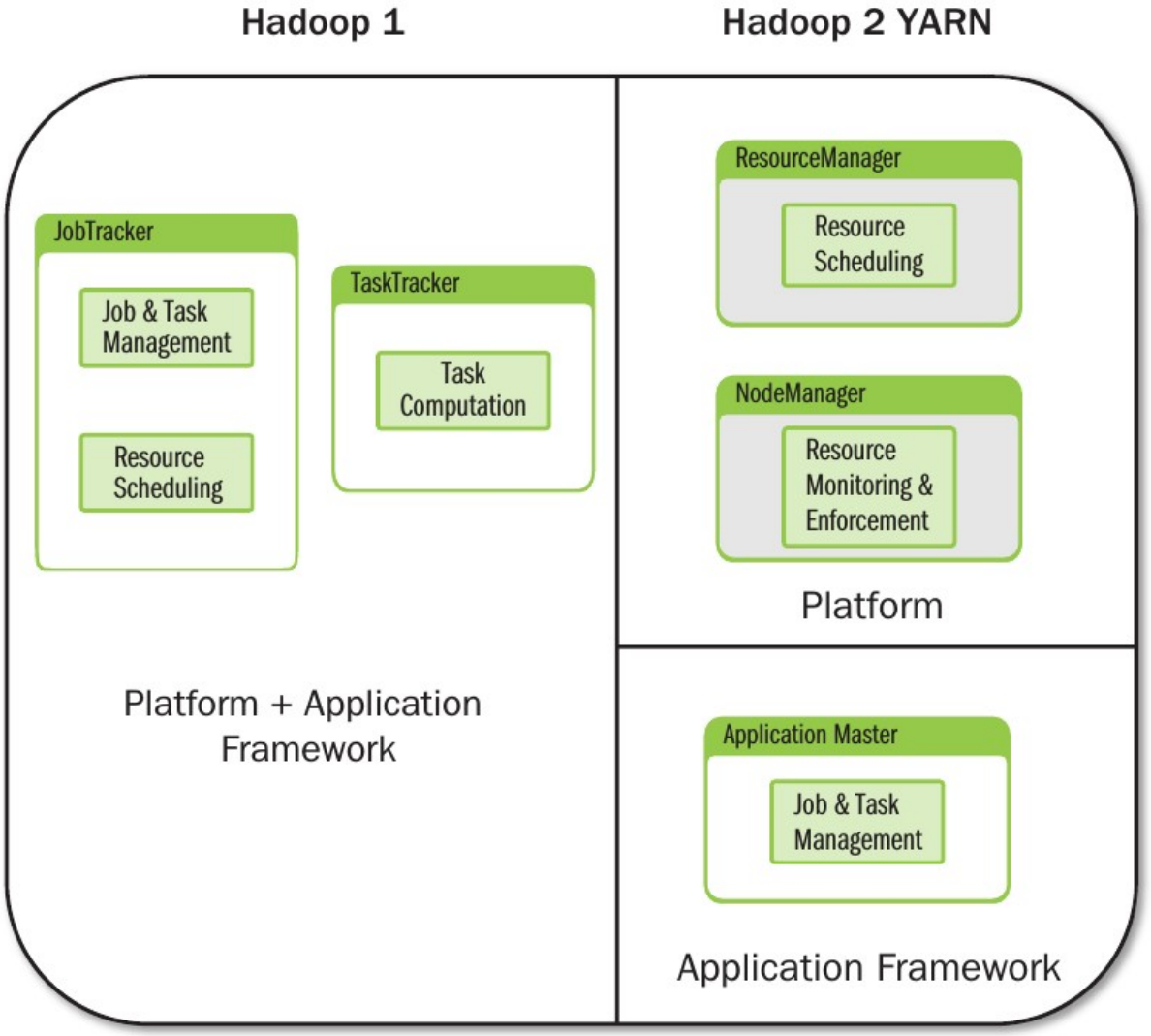
MapReduce Example

- Users tasks
  - Driver
  - Mapper,Reducer
  - Combiner
- Hadoop tasks
  - RecordReader
  - Shuffle
  - RecordWriter
- Cluster tasks
  - scheduling,
  - monitoring
  - resource management

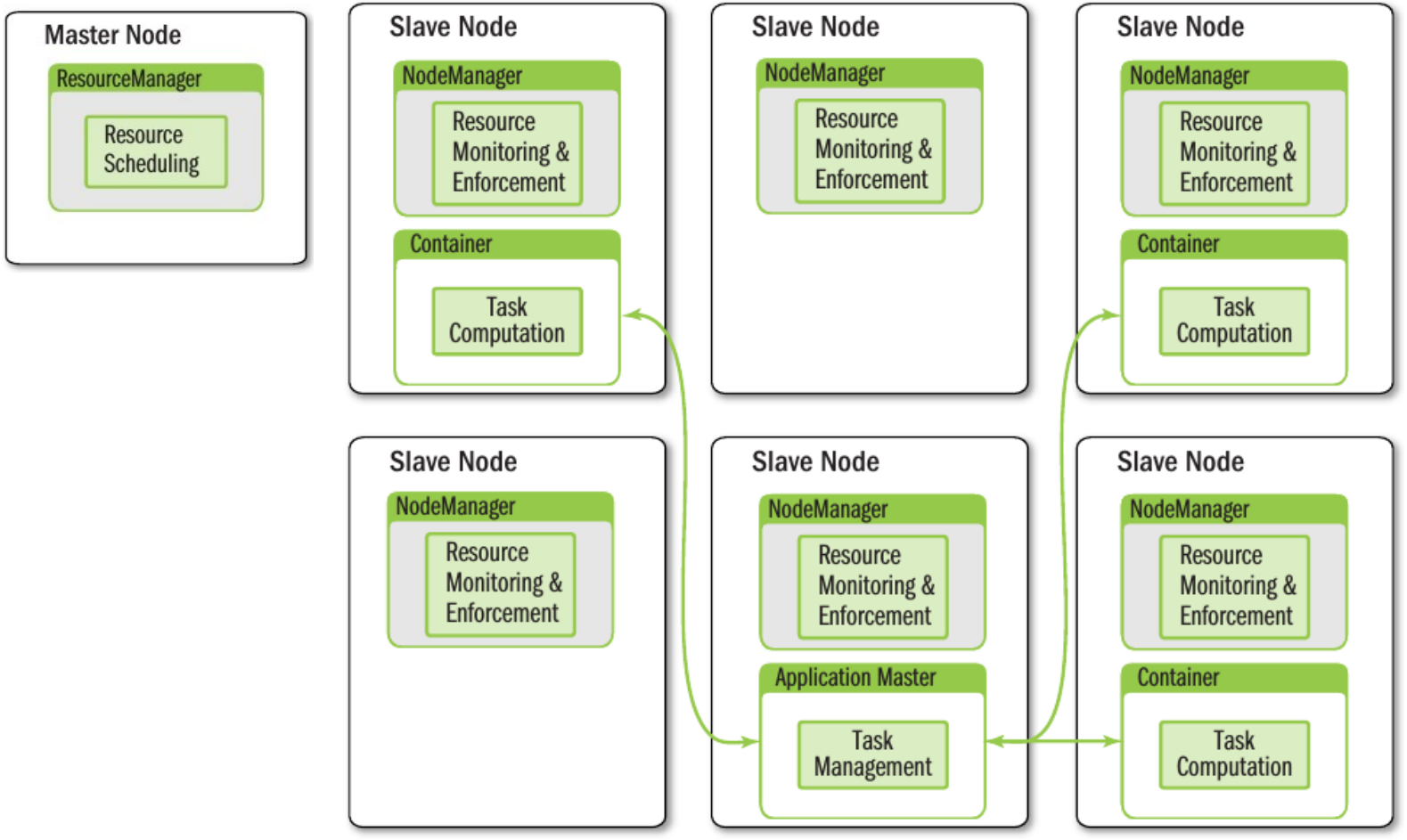




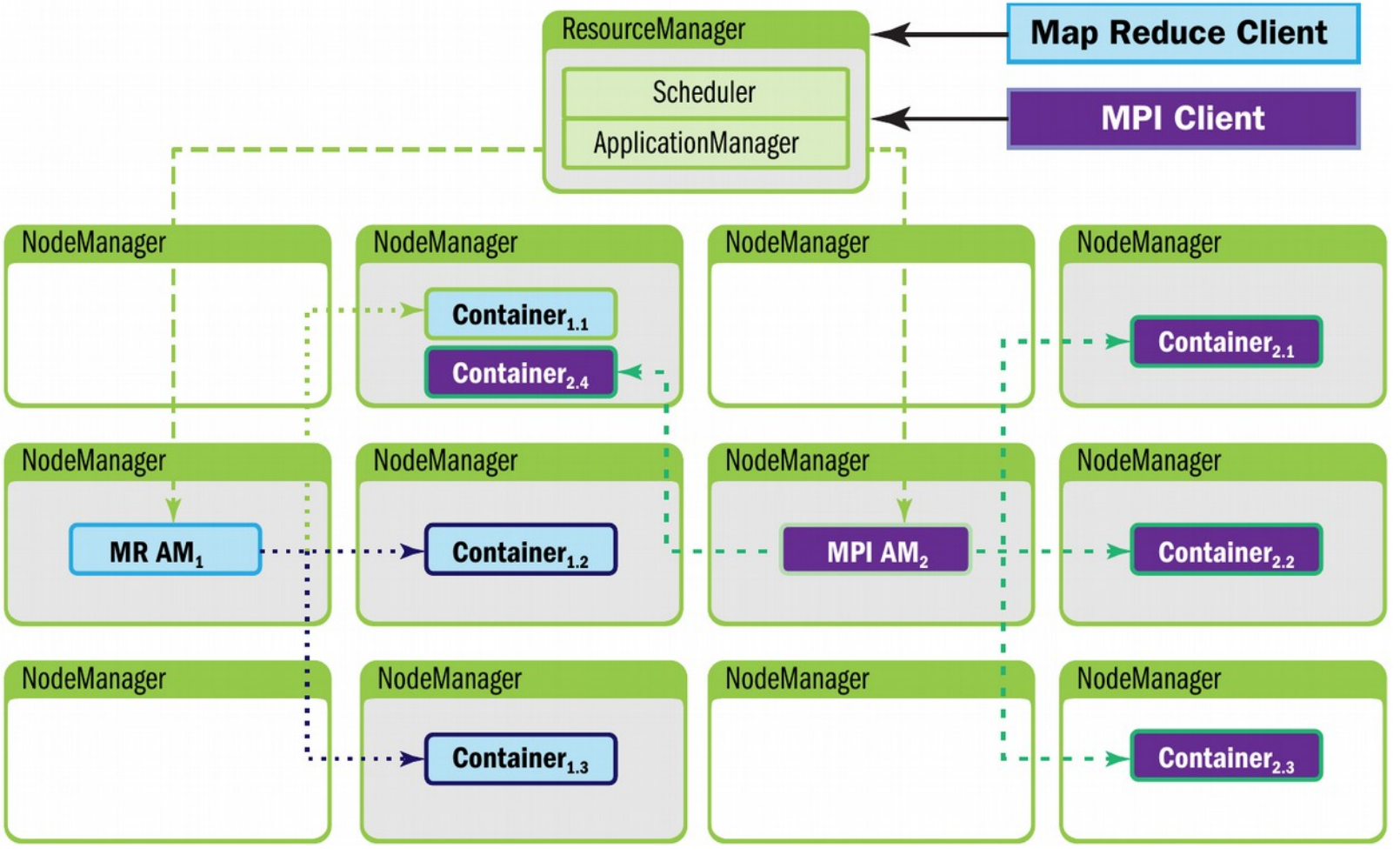
Hadoop 1 vs. Hadoop 2 YARN



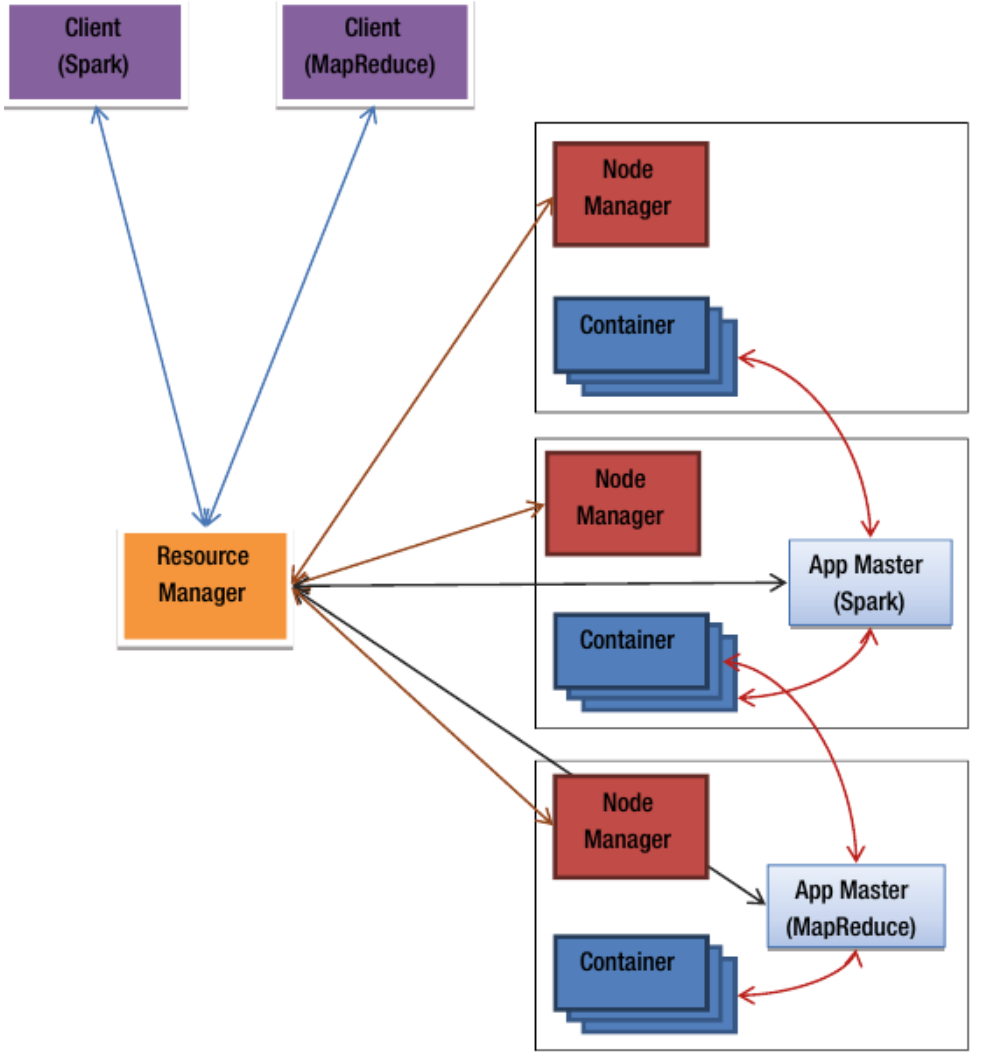
YARN Architectural Overview



YARN & App relationship



Example: Spark and MR tasks over YARN



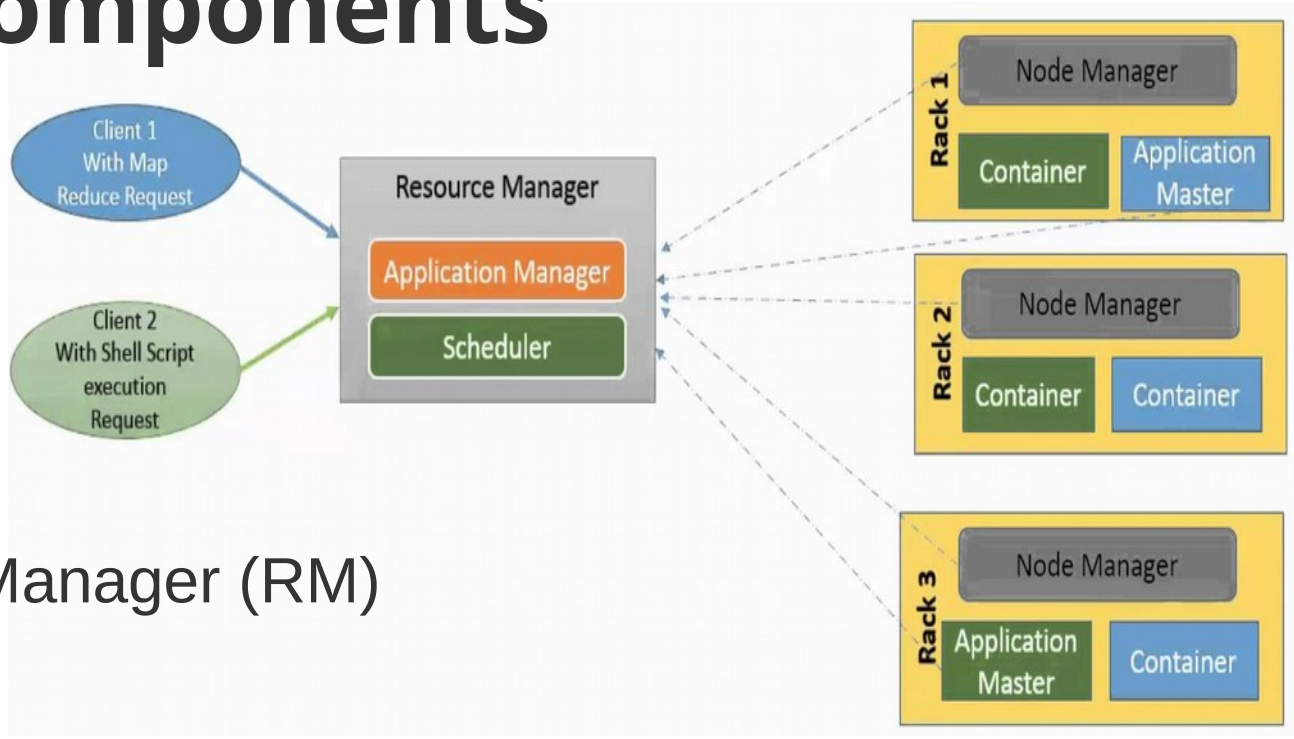
YARN Apps Categories

Type	Definition	Examples
Framework	Provides platform capabilities to enable data services	MapReduce, Spark, Tez, Twill, Reef, etc
Service	An application that runs continously	HBase, Memcached, Storm, etc
Job	A batch/iterative data processing task that runs on a service or framework	XML parsing MR job, Mahout K-means algo, etc
App	A temporal job or service submitted to YARN	HBase cluster, Map job, Reduce jobs, etc

YARN Platform Benefits

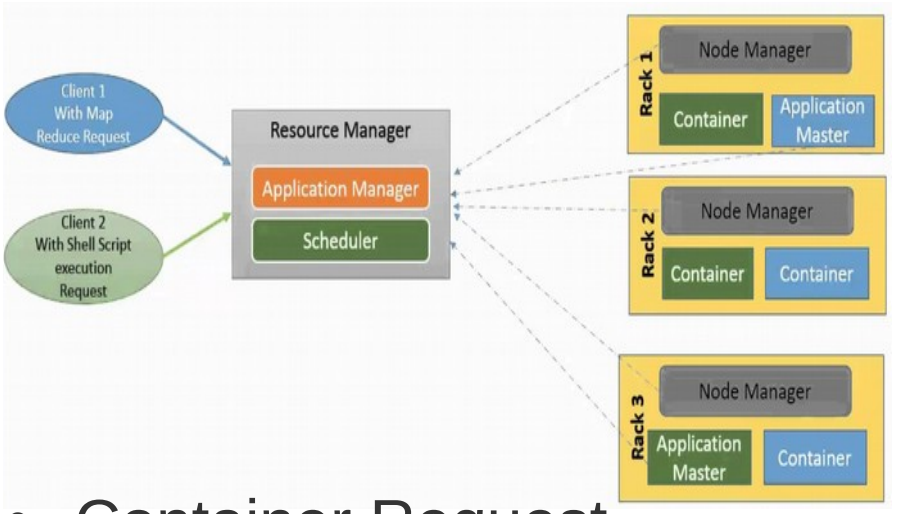
- Deployment
  - Provides a seamless vehicle to deploy a software to a cluster
- Fault Tolerance
  - Handles (detects, notifies, provides default actions) for HW / Network / OS / JVM failures
  - Provides plugins to define failure behaviors
- Scheduling
  - Applies chosen policies (FIFO, Fair, Capacity...)
  - Leverages HDFS to schedule application processing where the data lives

YARN Components



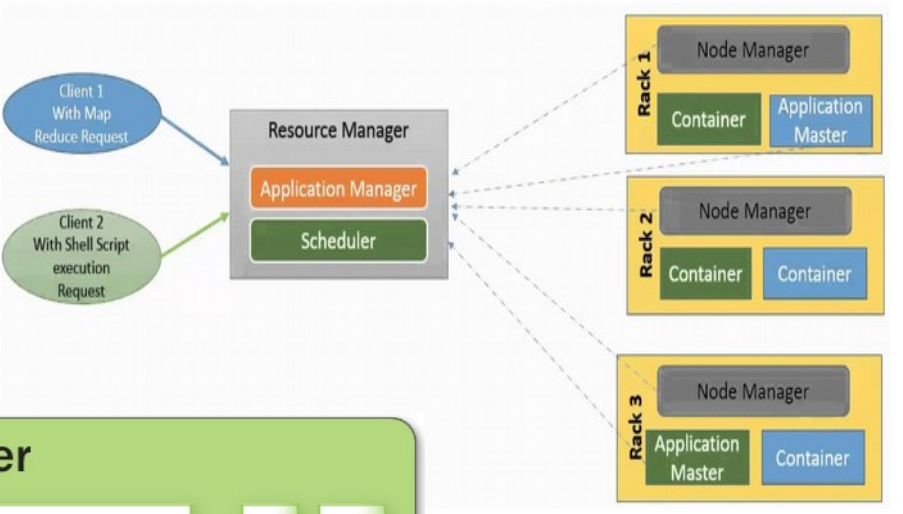
- Per-cluster
  - Resource Manager (RM)
- Per-node
  - Node Manager (NM)
- Per-application
  - Application Master (AM)
- Per-tasks
  - Containers

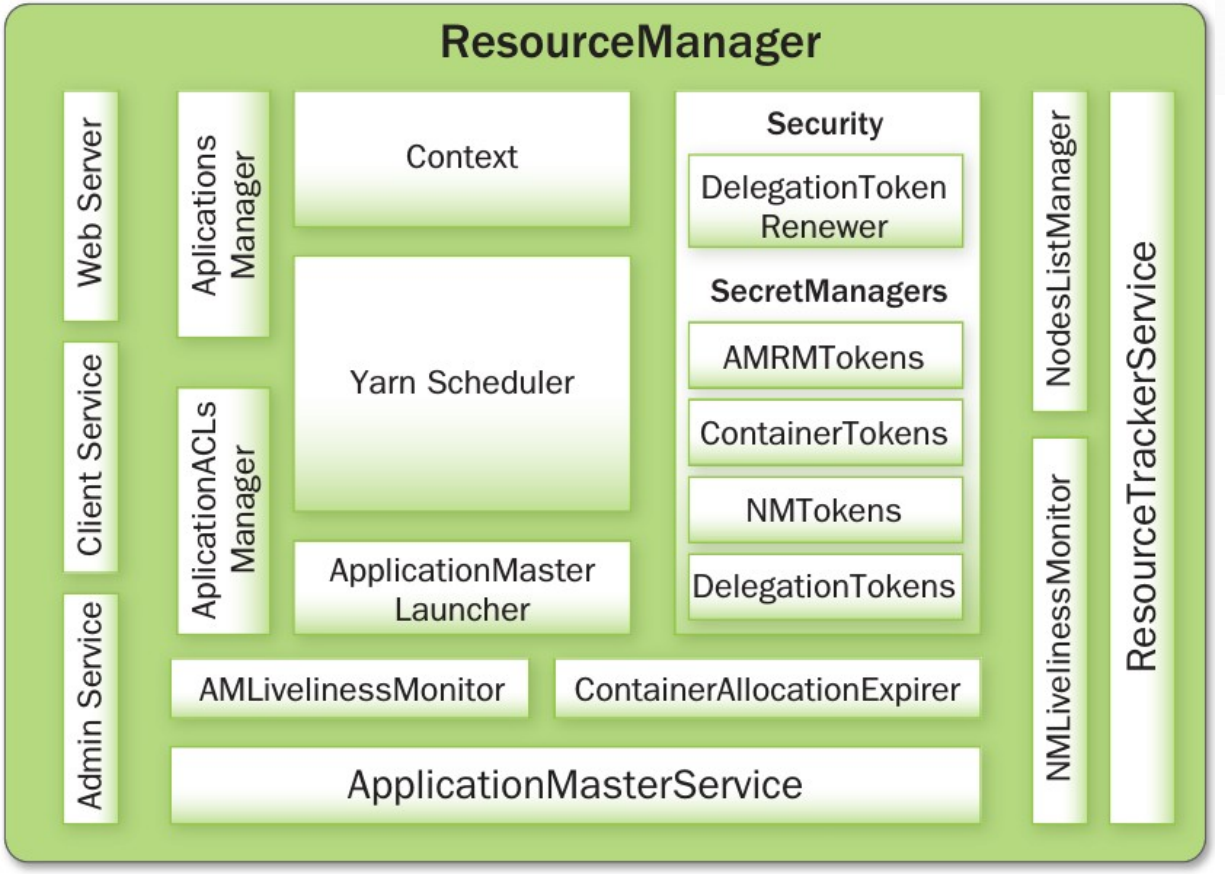
Container



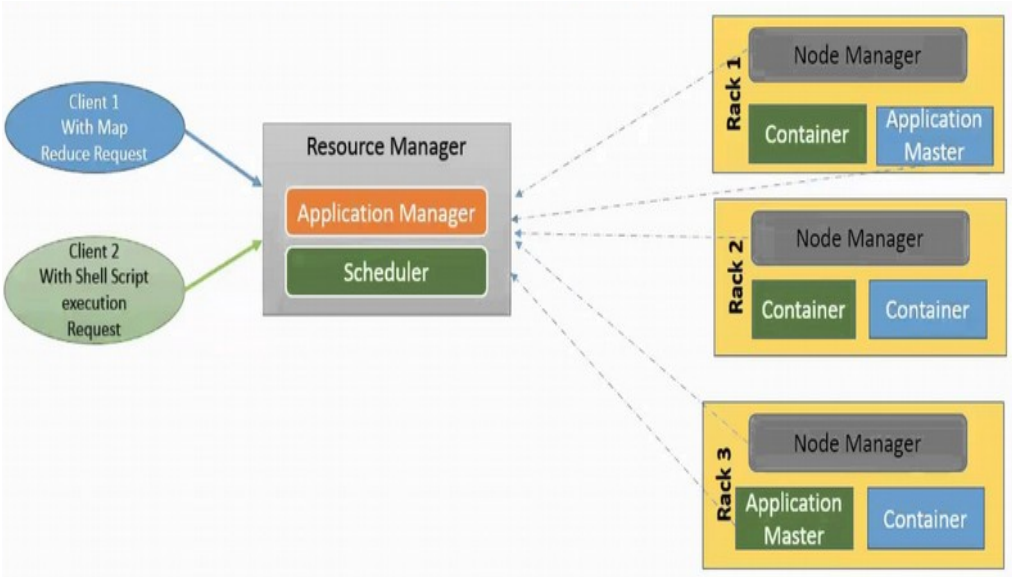
- Basic unit of allocation
- Fine-grained resource capability
  - Memory, CPU (cores), GPU, etc.
    - container_0 = 2 GB, 1 core
    - container_1 = 1 GB, 6 cores
- Container Request
  - capability, host, rack, priority, relax locality
- Container Launch Context
  - Resources needed to execute application
  - Environment variable
  - Command line to execute

Resource Manager Components



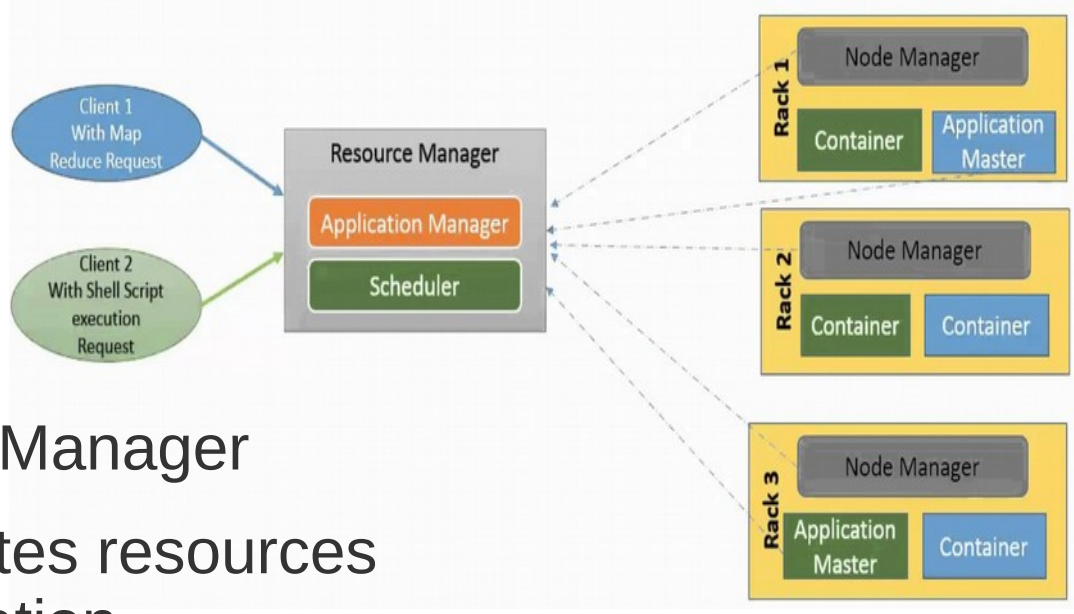


Application Manager



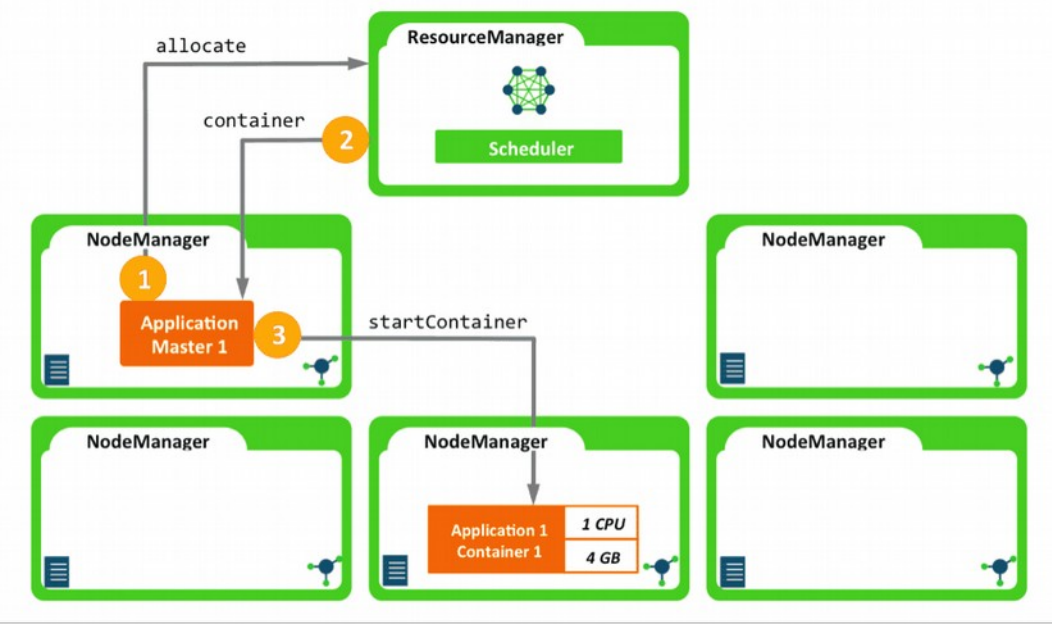
- Part of the ResourceManager
- Deals with job submission
- Negotiates the first container for the ApplicationMaster
- Start (or restart) the ApplicationMaster

Scheduler



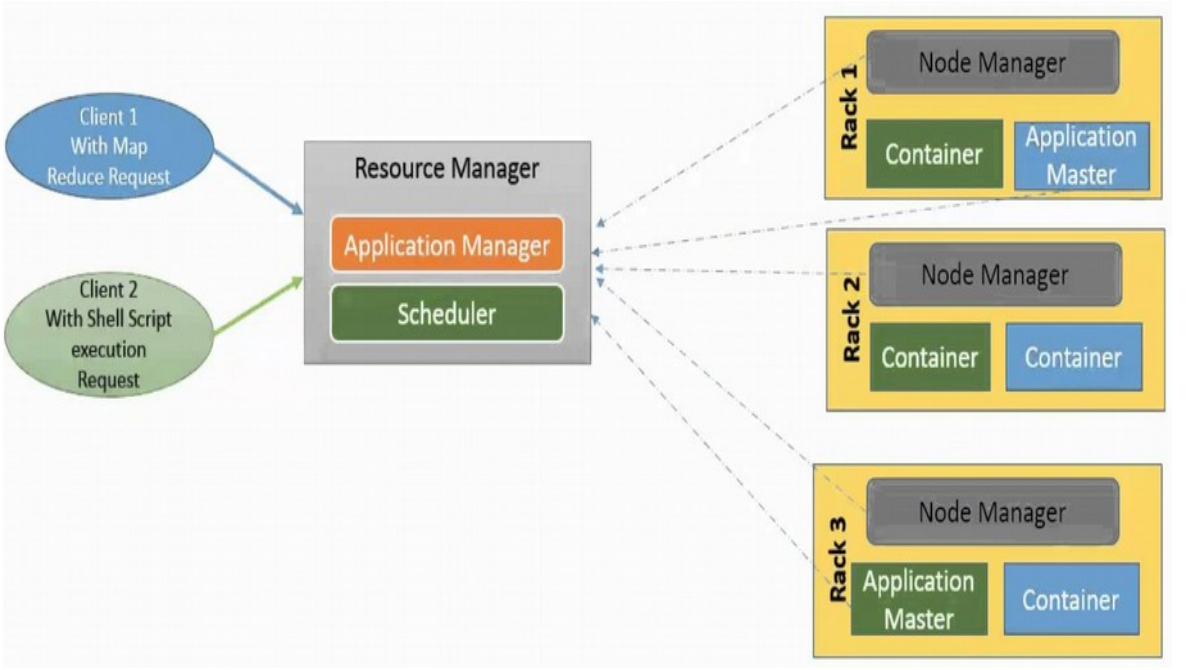
- Part of the ResourceManager
- Arbitrates and allocates resources among all the application
  - Based on resource requirements of application and containers informations
  - Pluggable policies
    - FIFO scheduler,
    - Capacity scheduler
    - Fair Scheduler...

Application Master



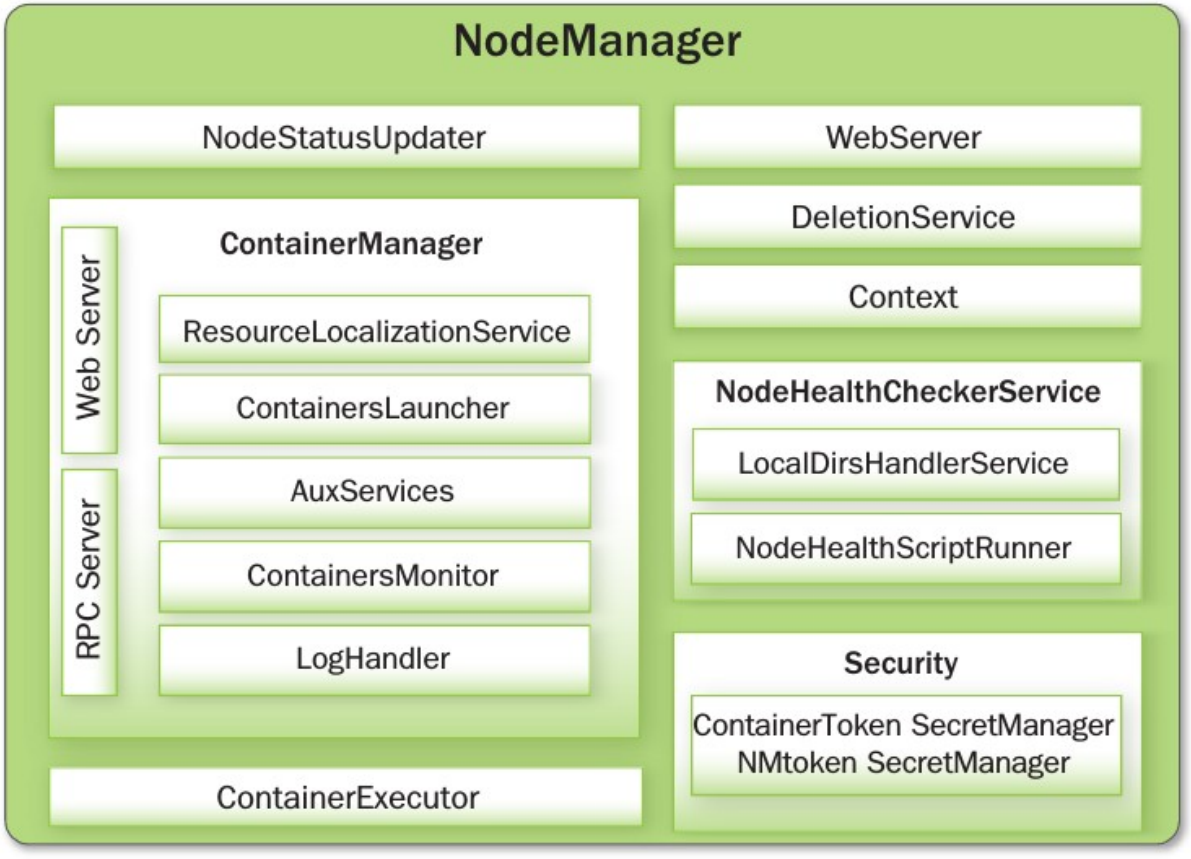
- Negotiates resource containers from the ResourceManager
- Launch the container by interacting with the NodeManager
- Coordinates the tasks of one application
- Tracks task status and manages fault tolerance

Node Manager

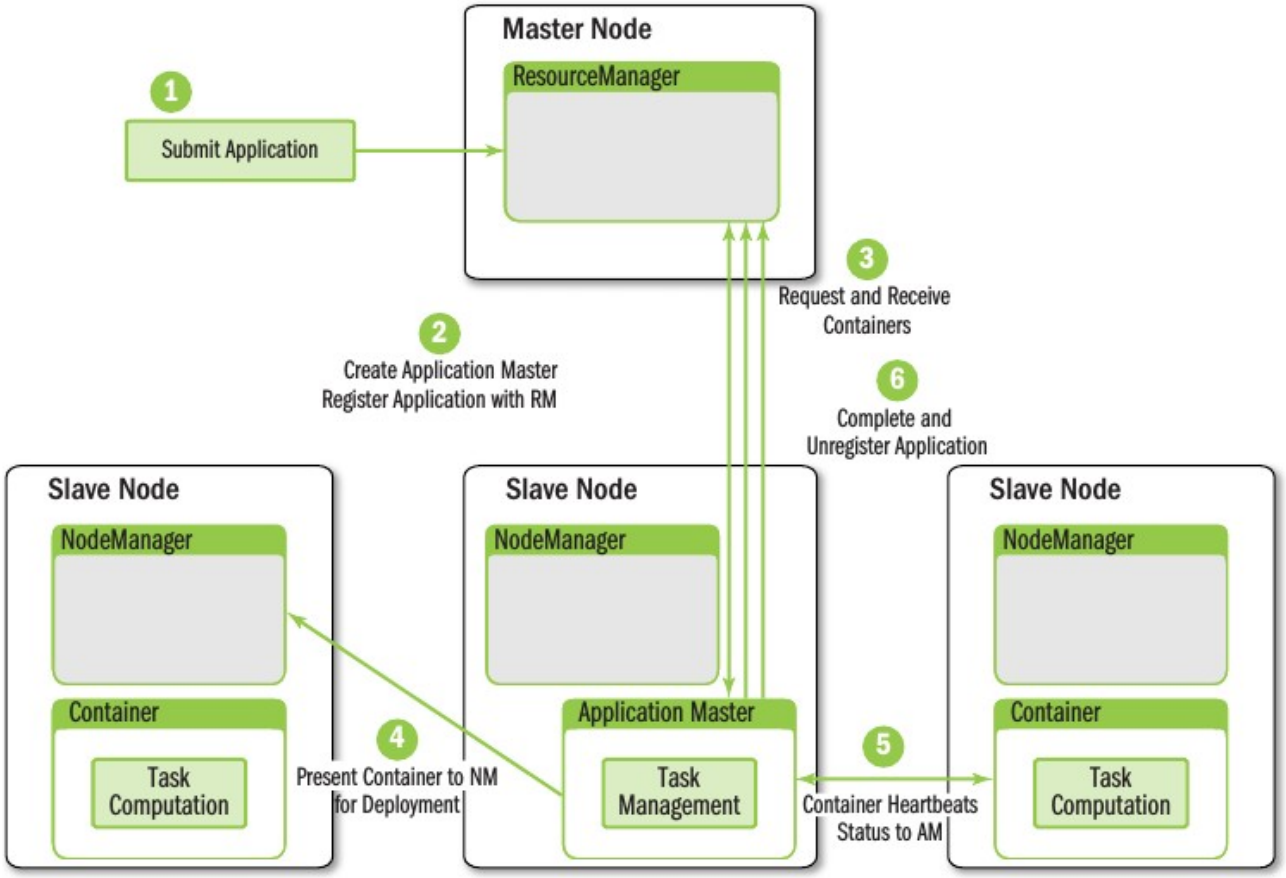


- Per-node agent
- Launches and manages node containers
- Monitors their resource usage
- Reports it to the Scheduler

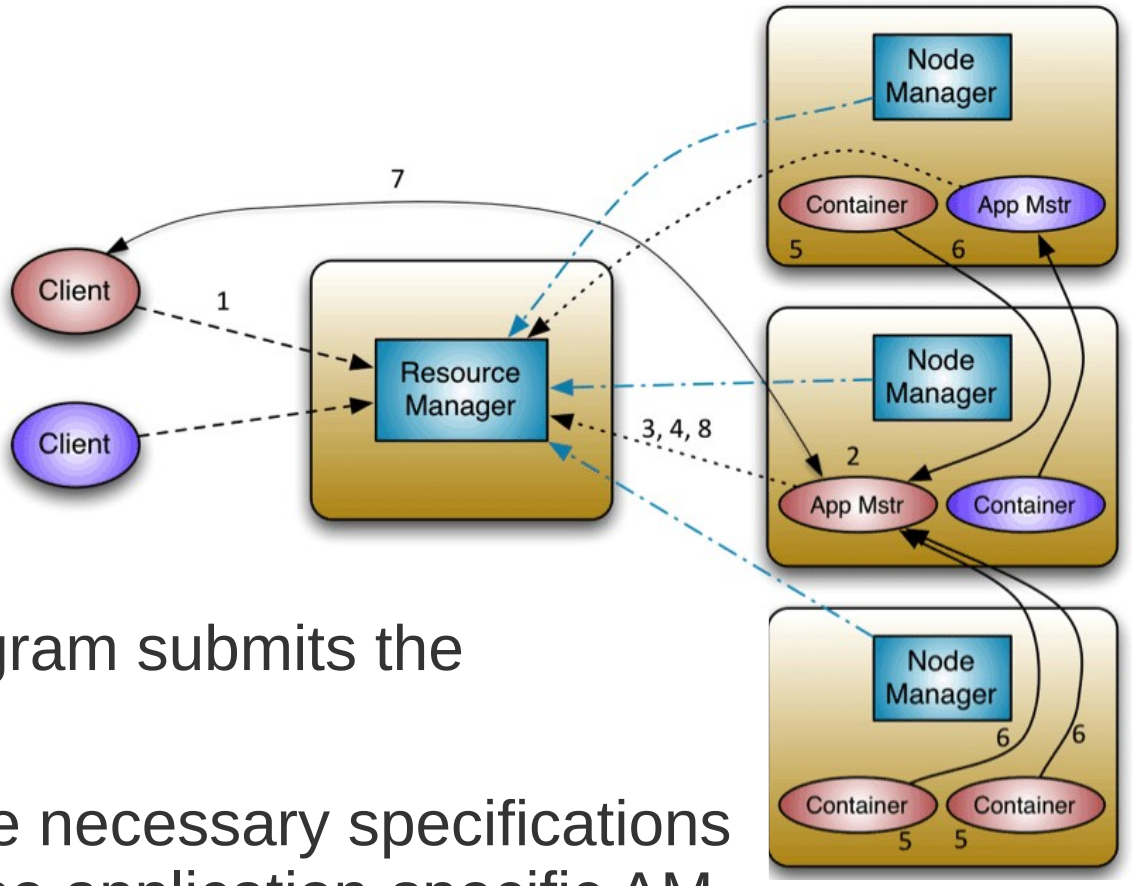
Node Manager Components



Application Data Flow

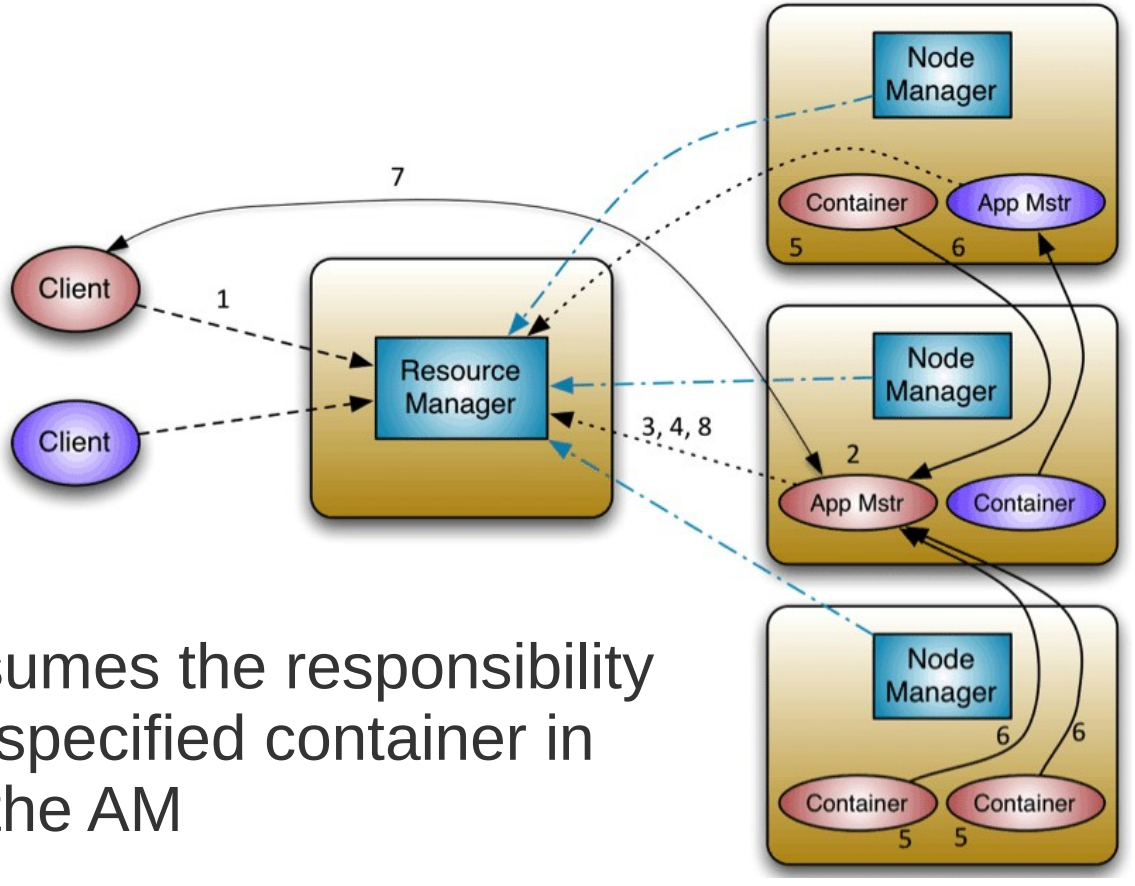


Walk Through Execution (1)



1. A client program submits the application
   - includes the necessary specifications to launch the application-specific AM itself.

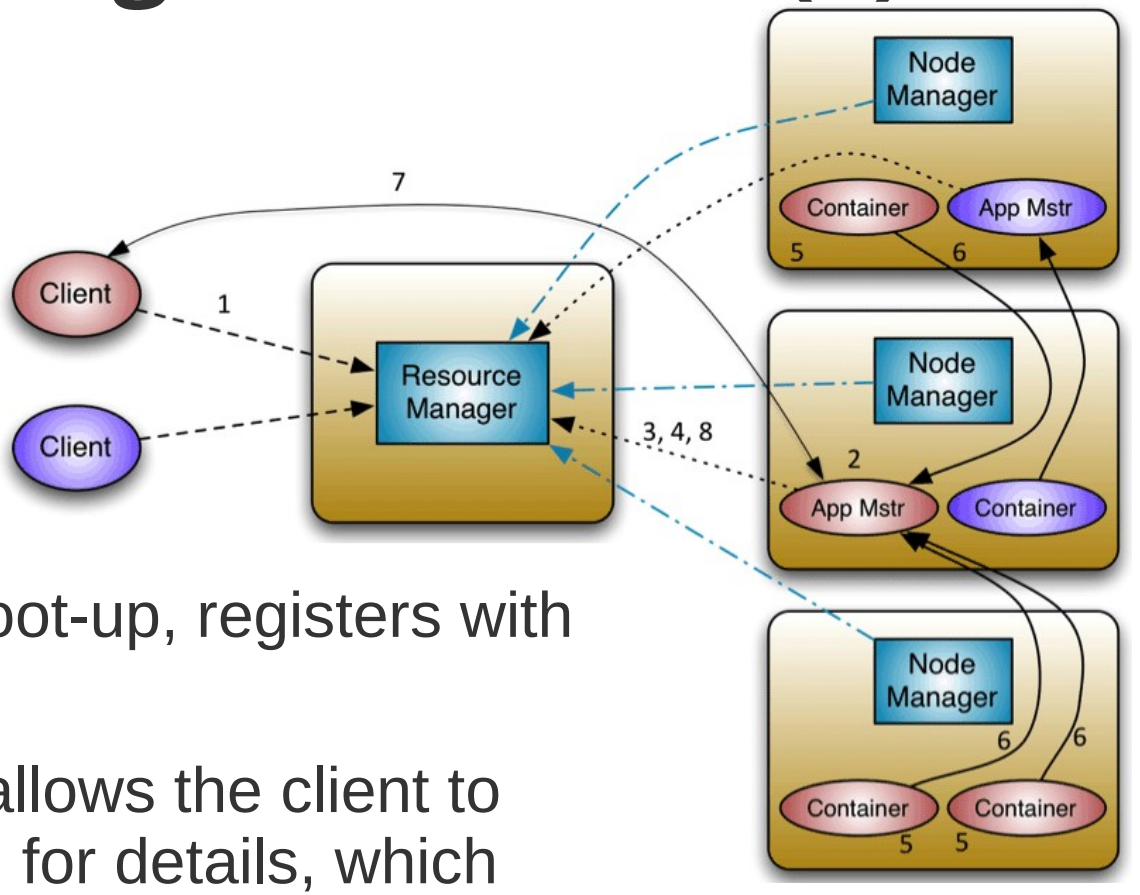
Walk Through Execution (2)



2. The RM assumes the responsibility to negotiate a specified container in which to start the AM

- then launches the AM.

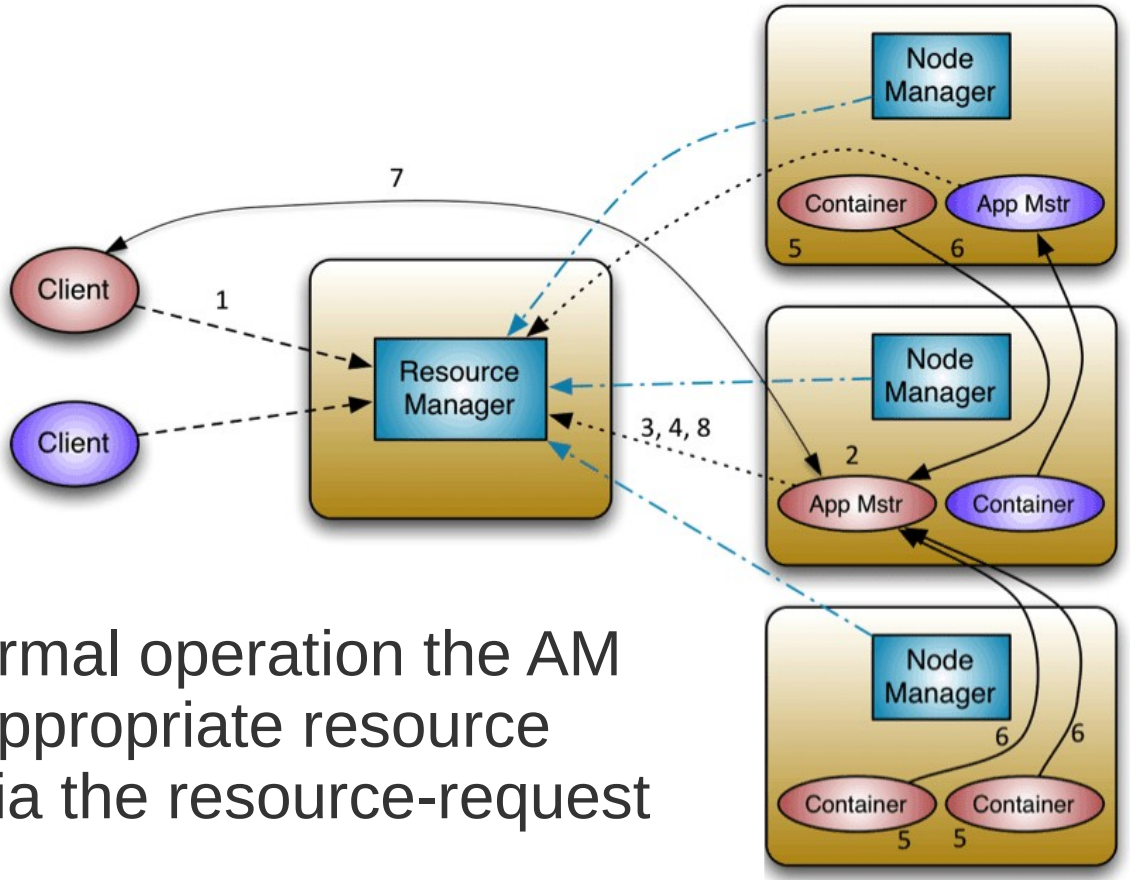
Walk Through Execution (3)



3. The AM, on boot-up, registers with the RM

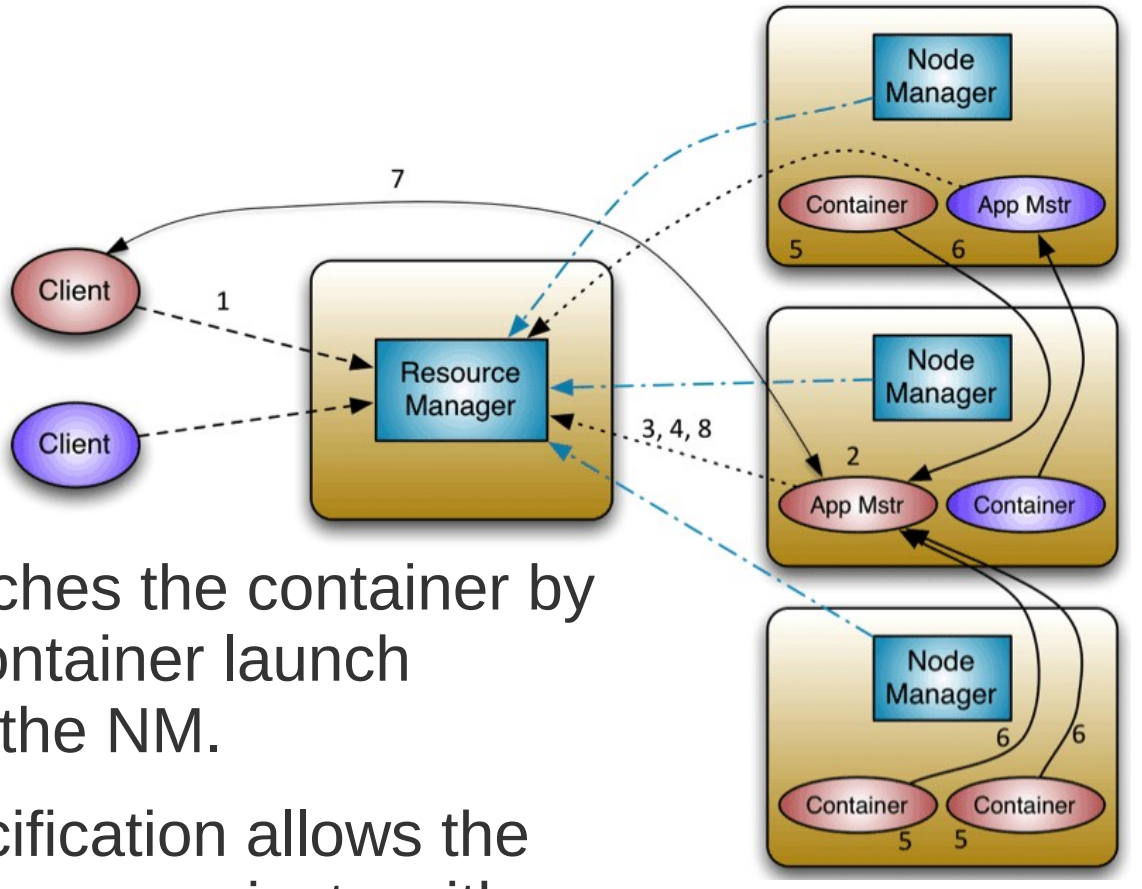
- Registration allows the client to query the RM for details, which allow it to directly communicate with its own AM.

Walk Through Execution (4)



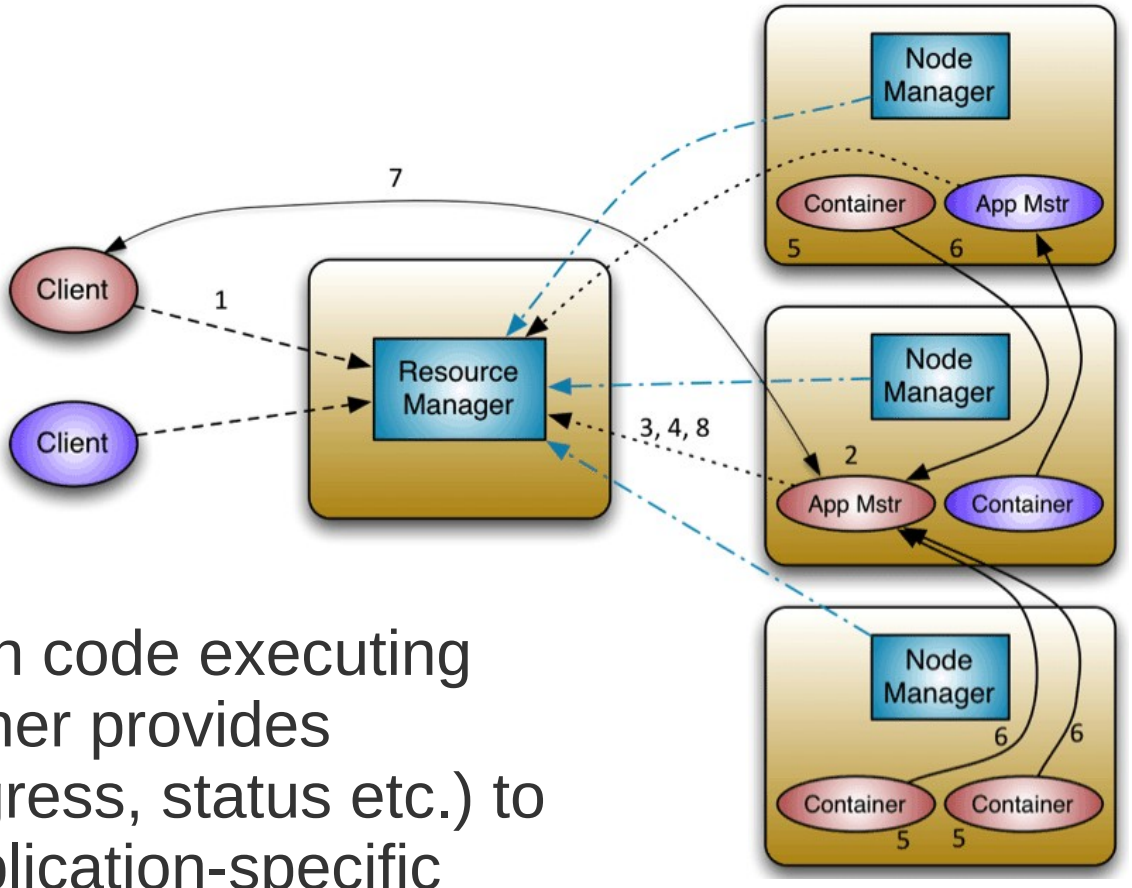
4. During normal operation the AM negotiates appropriate resource containers via the resource-request protocol.

Walk Through Execution (5)



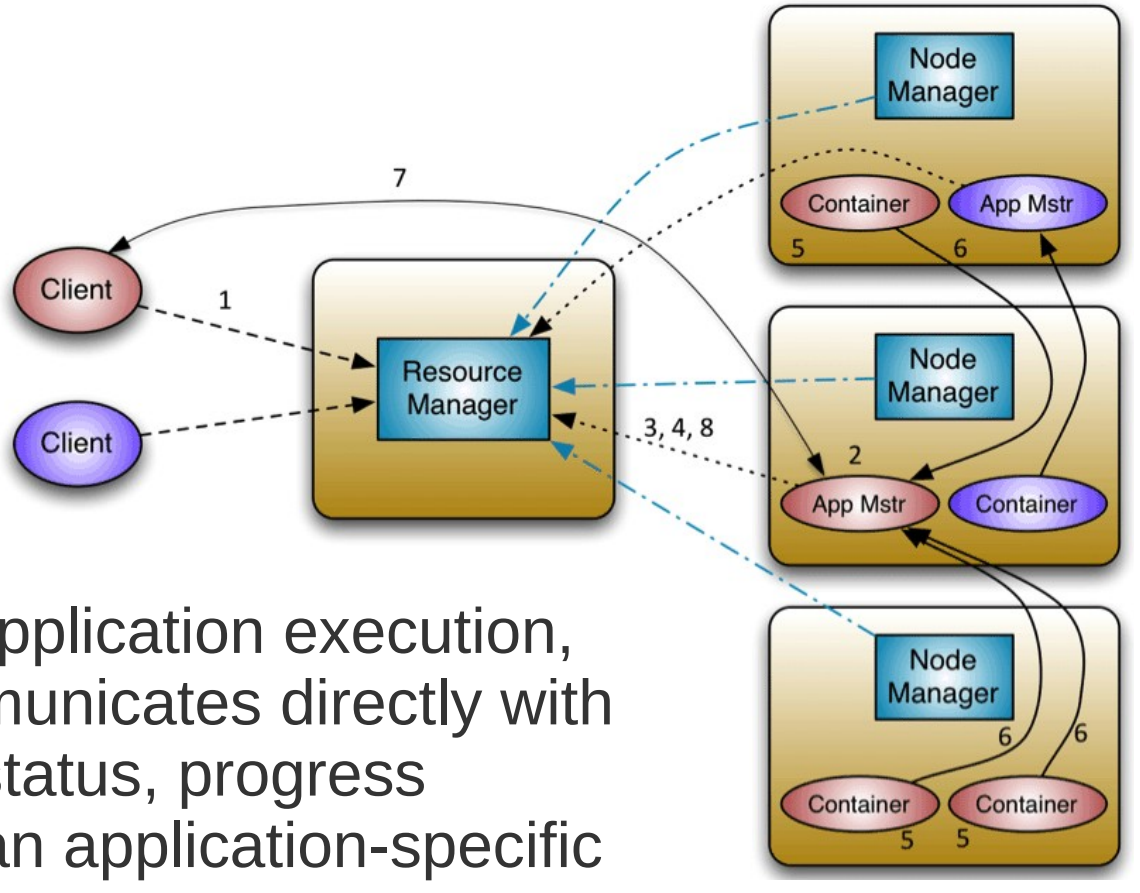
- 5. The AM launches the container by providing the container launch specification to the NM.
  - Launch specification allows the container to communicate with the AM itself.

Walk Through Execution (6)



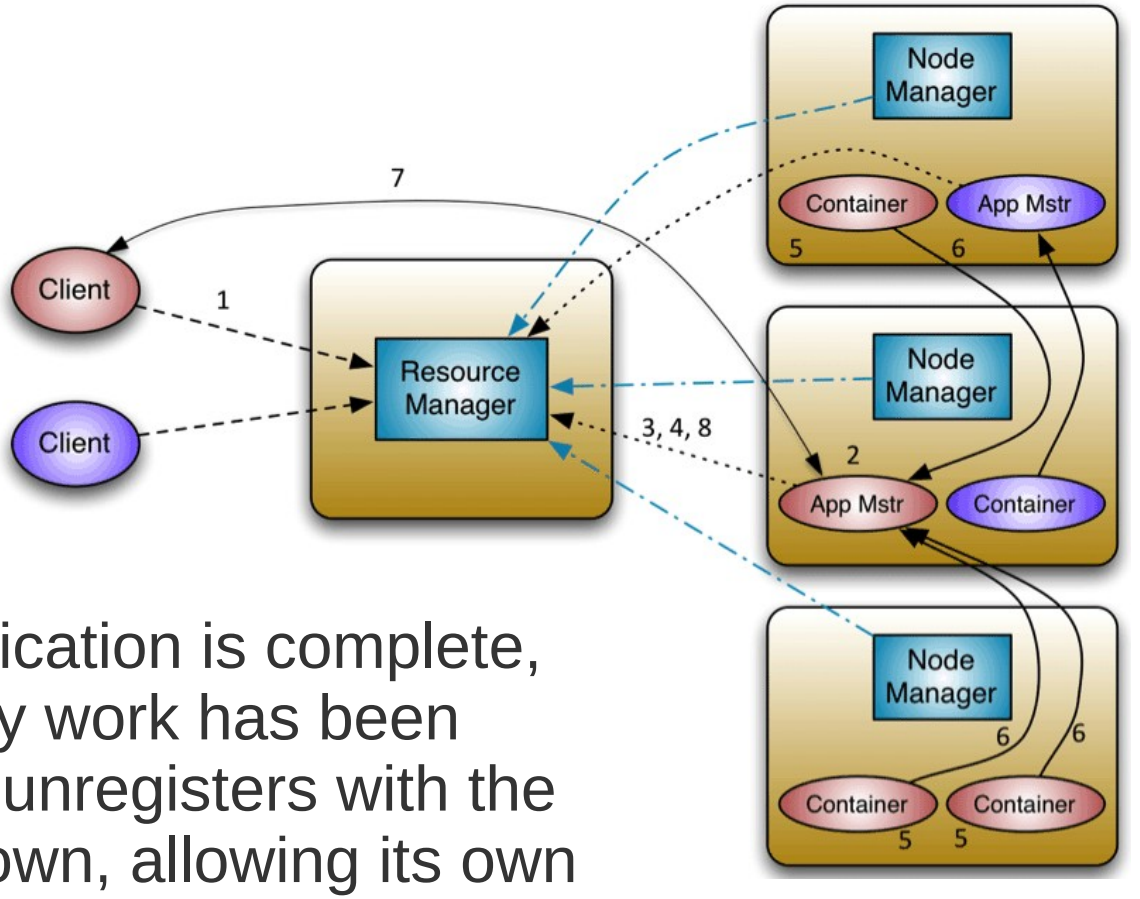
6. The application code executing within the container provides information (progress, status etc.) to its AM via an application-specific protocol.

Walk Through Execution (7)



- 7. During the application execution, the client communicates directly with the AM to get status, progress updates... via an application-specific protocol.

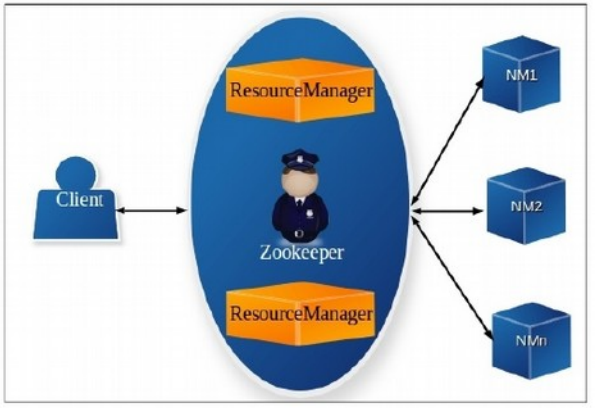
Walk Through Execution (8)



8. Once the application is complete, and all necessary work has been finished, the AM unregisters with the RM and shuts down, allowing its own container to be re-purposed

Faults Tolerance (1)

- ResourceManager failures
  - RM state shared between an active and a passive RM
  - Managed by a Zookeeper service



- ApplicationMaster failures
  - When the AM fails the RM starts another container with a new AM (and a new applicationID)
  - Recover state after restart is AM responsability
    - Should persist their states in an external location

Faults Tolerance (2)

- NodeManager failures
  - Detected by the RM using a heartbeat mechanism
  - The ResourceManager:
    - Removes this NM from the pool of available NM
    - kills all its containers
    - Reports the failure to all running AM
      - AM are responsible for reacting

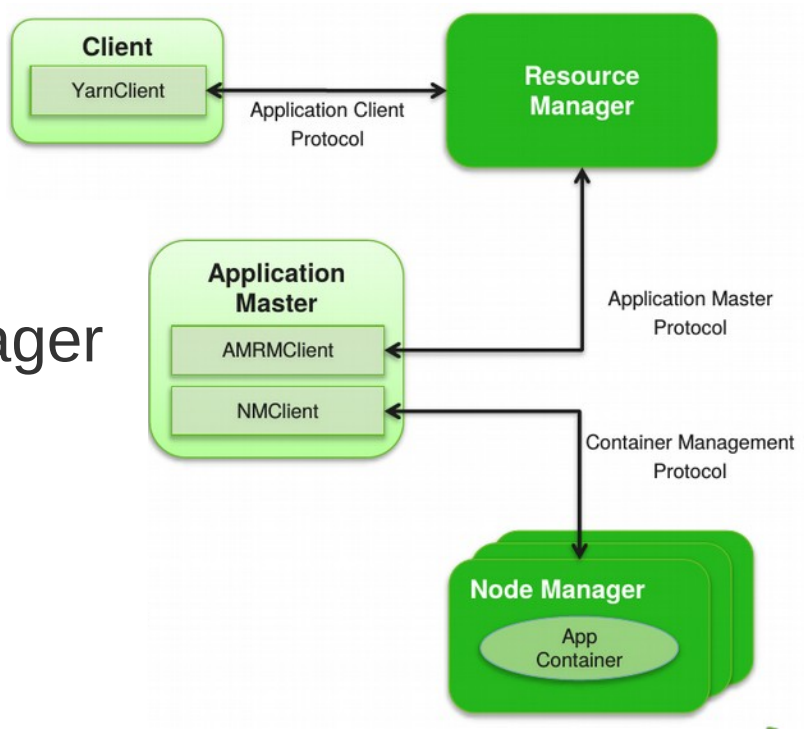
Faults Tolerance (3)

- Container failures
  - The AM is informed on the finishing of containers ; it receives the container status from the NM.
  - The AM interprets the exit status as a success or a failure and reacts accordingly
- Hardware failures
  - Automatically handled by the framework
    - Data replication across nodes/racks ensures data recovering and jobs resume

Implementation Guidelines

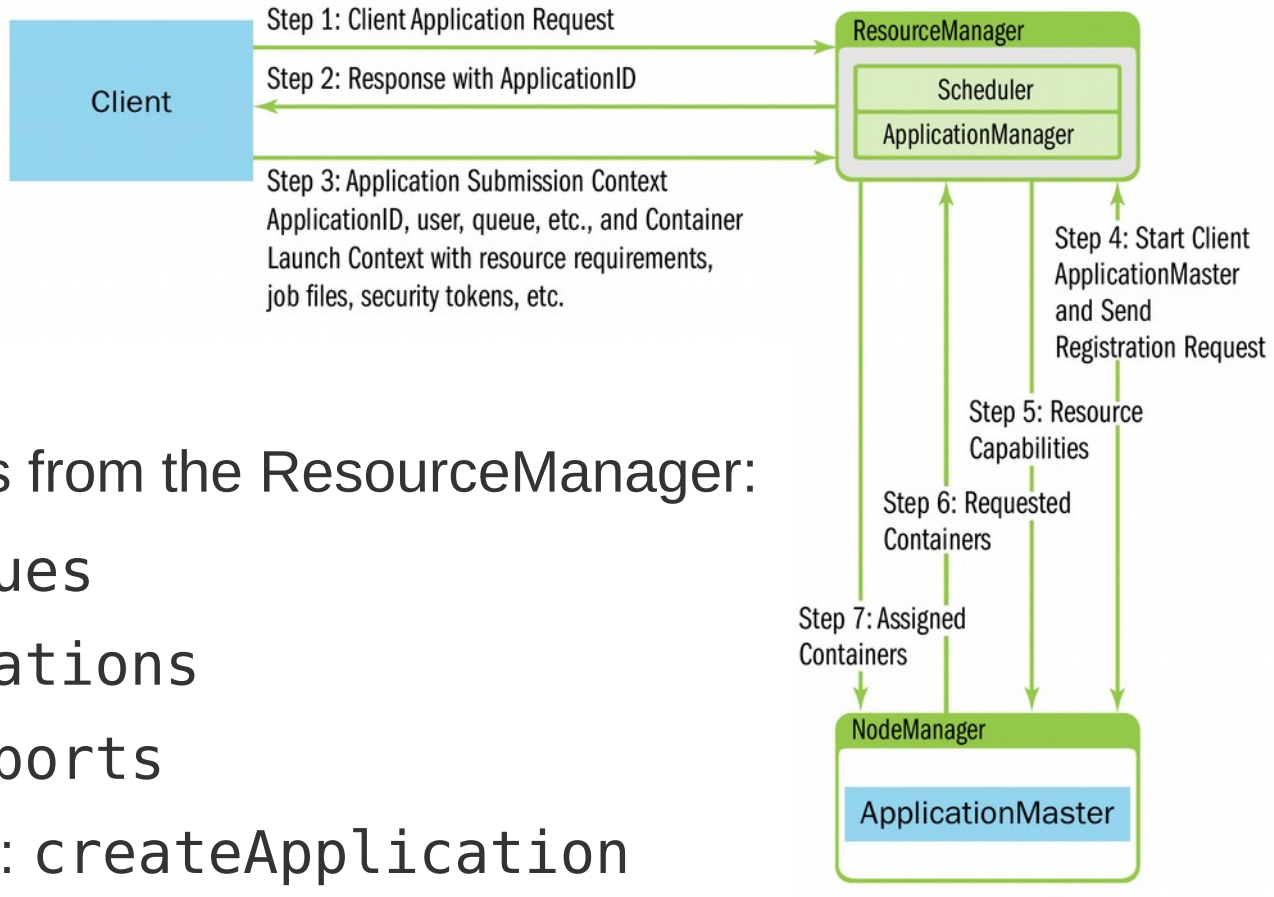
1) Write a Client to submit the application
2) Write an ApplicationMaster
   - Parent for all containers of the application
   - The ApplicationMaster negociates its containers from the ResourceManager
   - Restarted by the ResourceManager if required (unique `api.records.ApplicationAttemptId`)
3) Get containers and run whatever you want!

YARN API



- Application submission
  - Client to ResourceManager
- Container allocation
  - ApplicationMaster to ResourceManager
- Container launch
  - ApplicationMaster to NodeManager
- Packages under `org.apache.hadoop.yarn`
  - API starting point in the *cluster-pedago* wiki

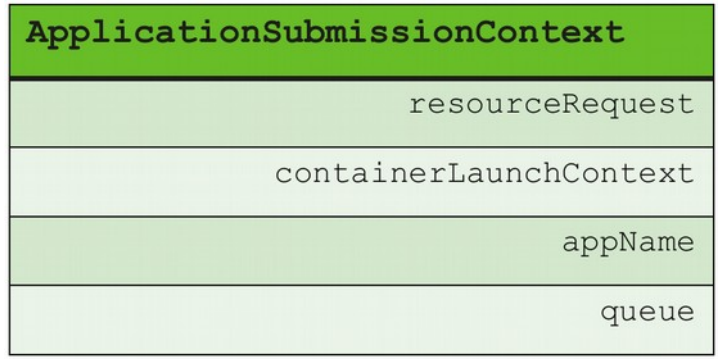
The Client



- `YarnClient`
  - Get informations from the ResourceManager:
    - `getAllQueues`
    - `getApplications`
    - `getNodeReports`
  - Creates the app: `createApplication`
  - Starts the app: `submitApplication`
    - An `ApplicationSubmissionContext` is needed to launch an AM on a node
  - Manipulates app: `killApplication...`

Specification of the AM

- `ApplicationSubmissionContext`
  - Context provided by the Client to the ResourceManager



- The ResourceManager is responsible for allocating and launching the ApplicationMaster `Container` (aka “container 0”)
- The AM will independently contact its allocated assigned node managers and provided them with a `ContainerLaunchContext`

Resource Request

- `api.records.ResourceRequest`
  - Fine-grained resource asked to the ResourceManager by the AM
  - For a specific amount of resources (CPU, memory, etc) on a specific machine or rack
    - Special value ANY or "*" for any host/rack

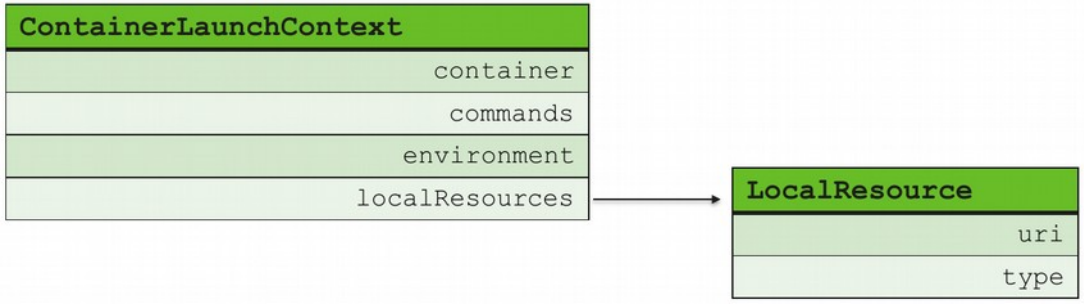
ResourceRequest
priority
resourceName
capability
numContainers

priority	capability	resourceName	numContainers
0	<2gb, 1 core>	host01	1
		rack0	1
		*	1
1	<4gb, 1 core>	*	1

Container

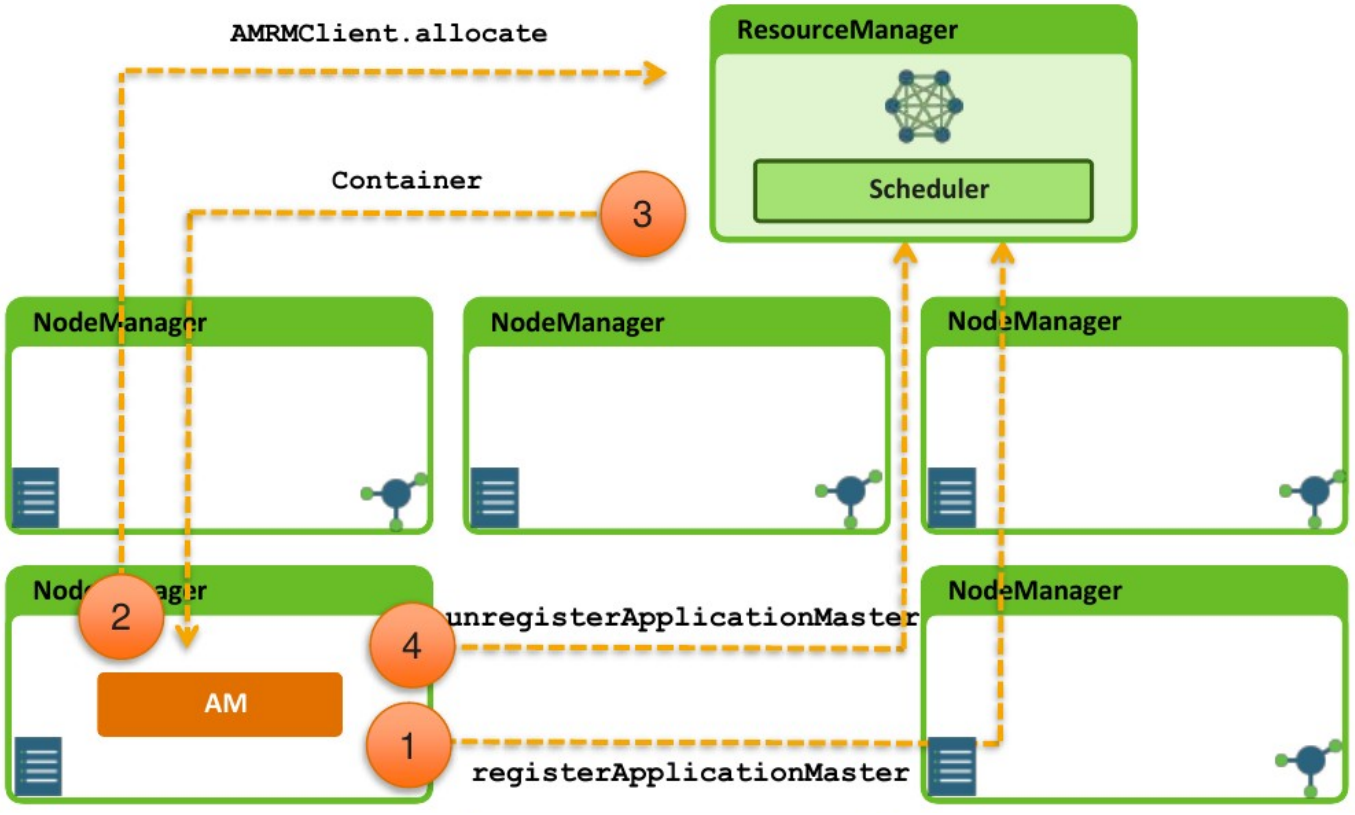
- `api.records.Container`
- Result of a `ResourceRequest` provided by the ResourceManager to the ApplicationMaster
- Includes :
  - `ContainerId` for the container
  - `NodeId` of the node on which it is allocated.
  - HTTP uri of the node.
  - `Resource` allocated to the container.
  - `Priority` at which the container was allocated.
  - `Token` of the container for authenticity

Resource Usage



- `ContainerLaunchContext`
  - Context provided by the ApplicationMaster to a NodeManager to launch the `Container`
- `LocalResource`
  - Specify files (binary, configuration and application files…) or library (jar file...) required to launch a container
  - Some resources can be localized (copied) onto the local file system by the NM prior to launch a container
  - Types are `FILE`, `ARCHIVE`, `PATTERN`

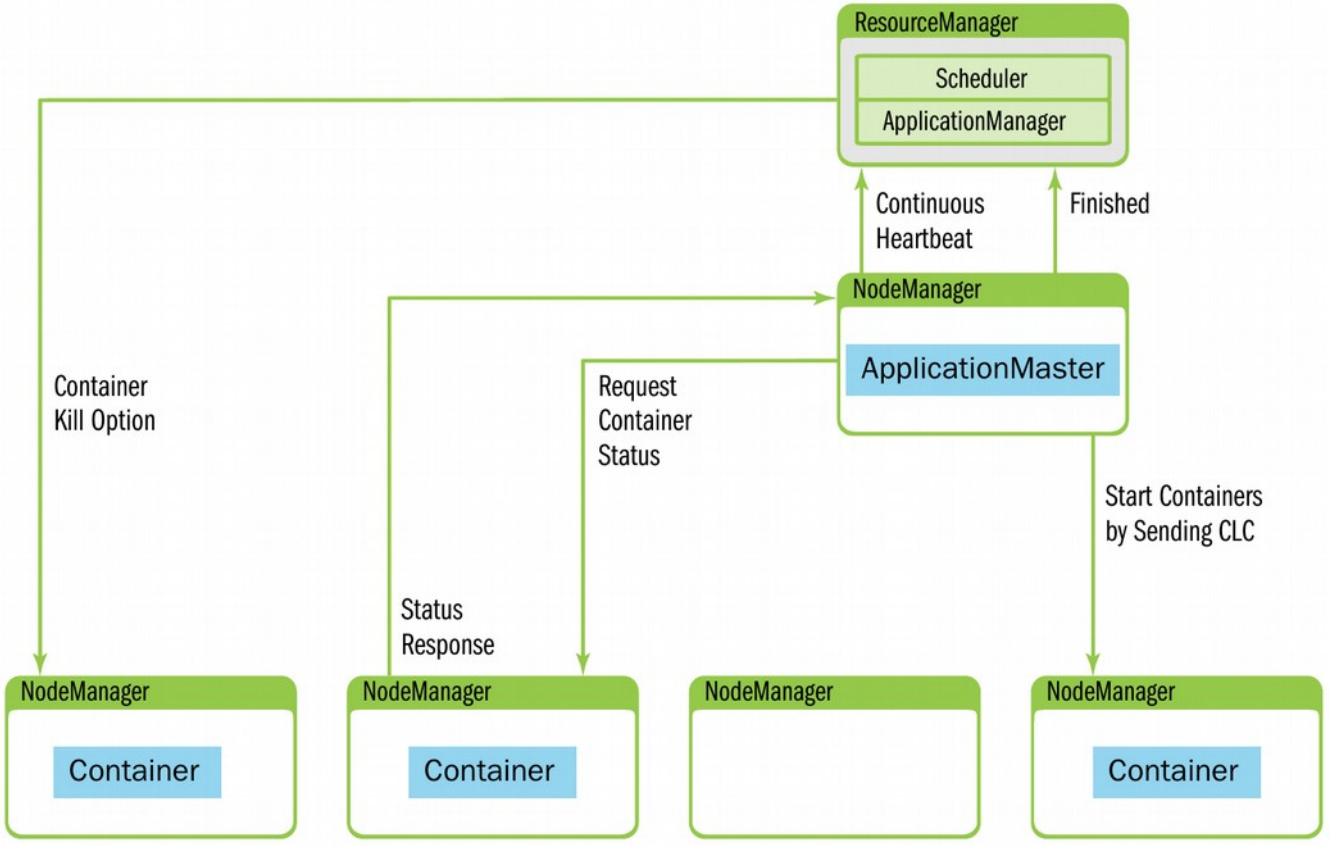
AM to RM



- Synchronous or Asynchronous
  - `AMRMClient` vs `AMRMClientAsync`

- AM un/registering
- Resource negotiation
- Helpers for information

AM to NM



- Synchronous or Asynchronous
  - `AMNMClient` vs `AMNMClientAsync`
- `start/stop/getContainer`
- Example: Map and Reduce tasks

A Simple YARN Application

- Run the same Unix command on each node
  - `yarn jar dshell.jar dshell.Client date 3 /apps/dshell.jar`
  - Do not forget to copy `dshell.jar` on HDFS
- Yarn Client (`dshell.Client`):
  - Prepares and launches the AM as a java program (`java dshell.AppMaster`)
  - Probes the application states report
- ApplicationMaster (`dshell.AppMaster`):
  - Obtains, prepares and run `N` containers

YARN CLI

- General help

```
yarn -h
```

- Submit a job to the RM

```
yarn jar <myapp.jar> <myMainClass> <params>
```

- List all submitted jobs

```
yarn application -list -appStates ALL
```

- Kill a job

```
yarn application -kill <myapp ID>
```

- Print the stdout logs of a job

```
yarn logs -applicationId <myapp ID>
          --log_files stdout
```

YARN Alternatives

- Mesos
  - Originally at UCB, now open source under the Apache
  - http://mesos.apache.org/
  - Highly-available and fault-tolerant OS kernel for cluster
- Omega
  - Google's next generation cluster management system
  - New parallel scheduler architecture
  - http://research.google.com/pubs/pub41684.html
- Corona
  - Work from Facebook, now hosted on GitHub
  - Rewrite and specialization of the Hadoop scheduler for fat client
  - https://github.com/facebookarchive/hadoop-20/tree/master/src/contrib/corona