Applying ML to Big Data using Hadoop and Spark Ecosystem

Map Reduce, YARN
Spark

Dr. Prasad M Deshpande

Slides adapted from Manoj Duse

Recap from Last Session

- Big Data Use Cases
- What are two important components of Hadoop?
- Big Picture of various Apache Big Data projects [Hadoop Ecosystem]
- Hadoop Vendors Distributions
- Composition of Data Centre
- HDFS : which two daemons? Who does what? Fault-tolerance?
- Scalable Cluster ? HA?

Applying ML to Big Data using Hadoop and Spark Ecosystem

1. Foundations & Distributed Storage
2. Resource Management & Parallel Processing

Agenda

- Map Reduce
- Resource Management : YARN
- Spark
- Lab

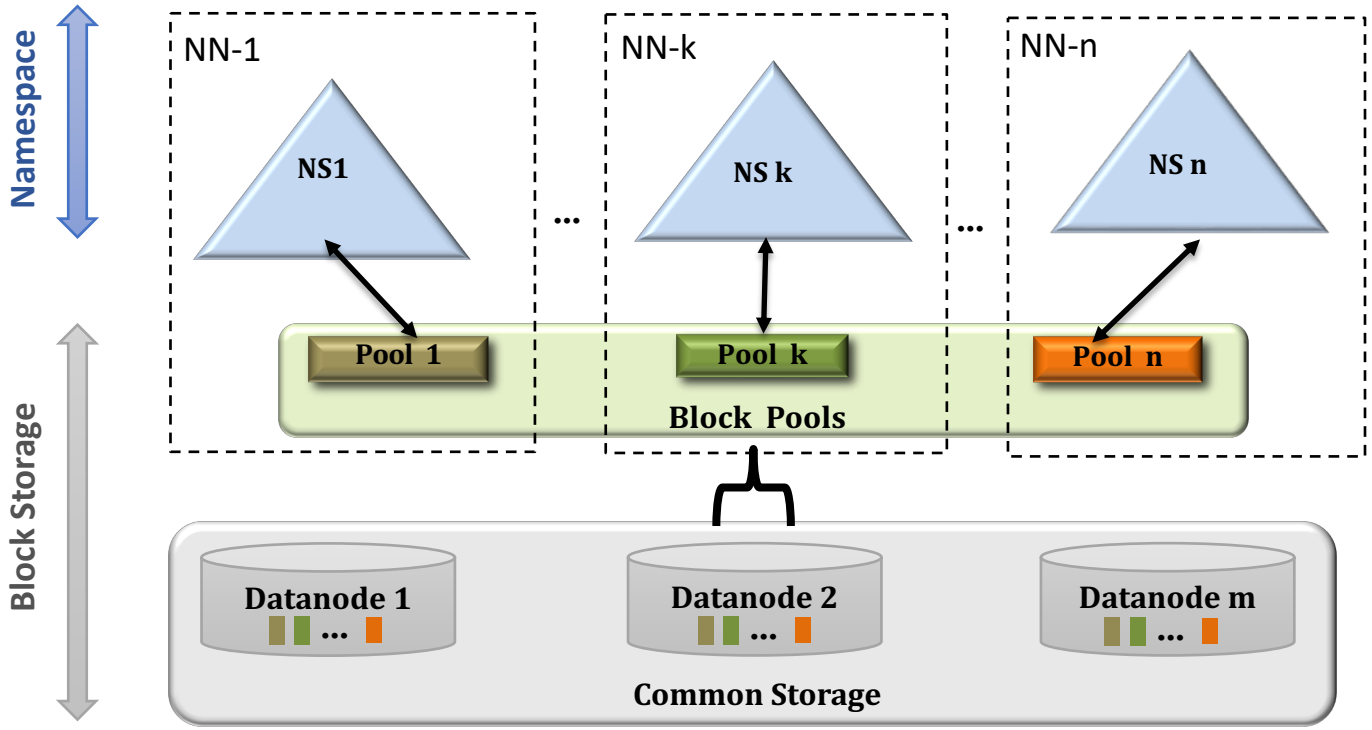
Metadata Size : Memory Requirement

- Rule of thumb

- 1000 MB per Million Blocks of file storage

- 24 TB Disk, 200 node cluster
- 200 * 24,000,000 MB / [128MB * 3] ~ 12 million blocks

HDFS 2.0: Name Node Federation Elaborated

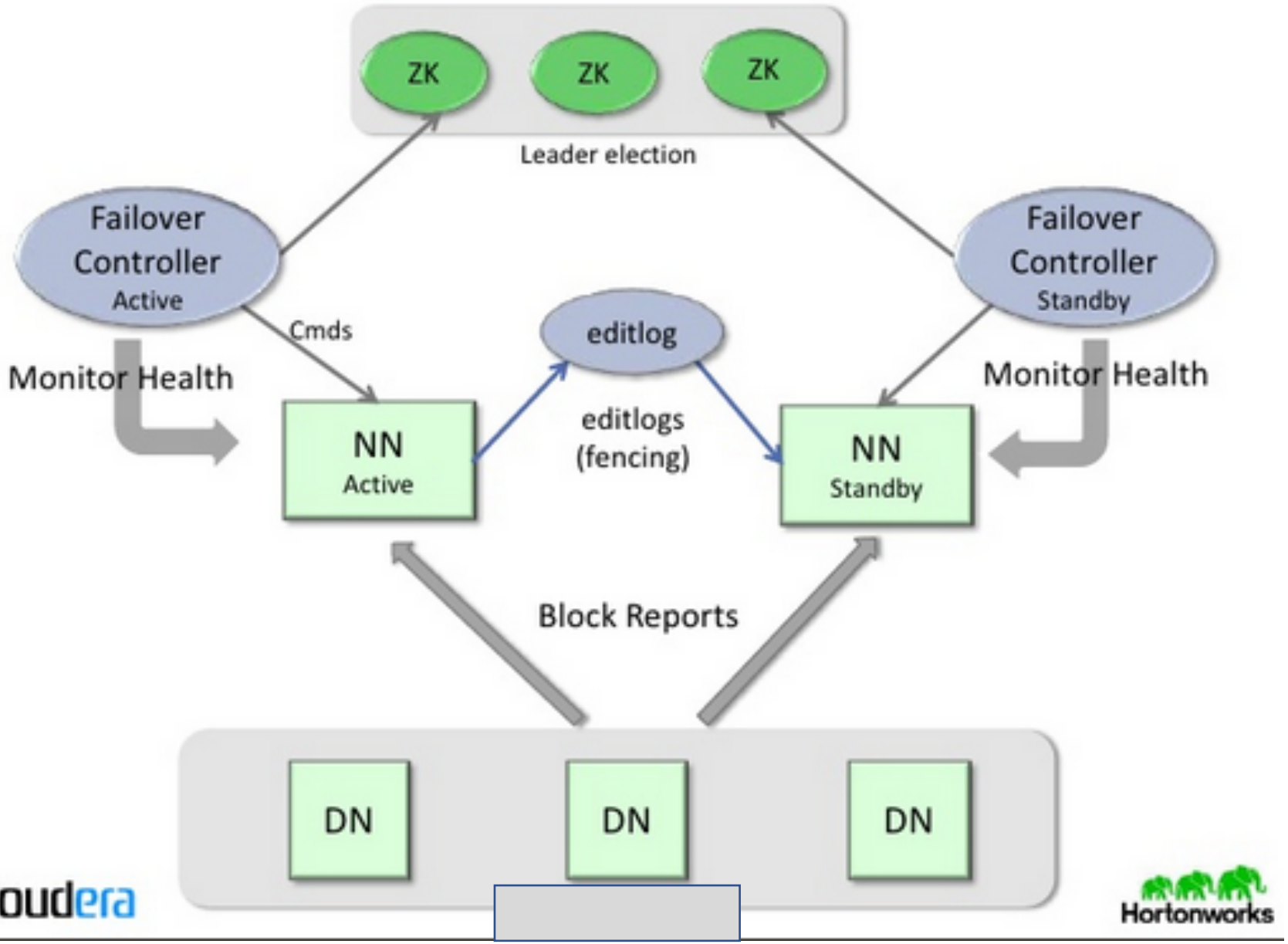


- Multiple **independent** Namenodes and Namespace Volumes in a cluster
  - Namespace Volume = Namespace + Block Pool
  - Set of blocks for a Namespace Volume is called a ***Block Pool***
  - DNs store blocks for all the Namespace Volumes – no partitioning

What benefits come from “Federation”?

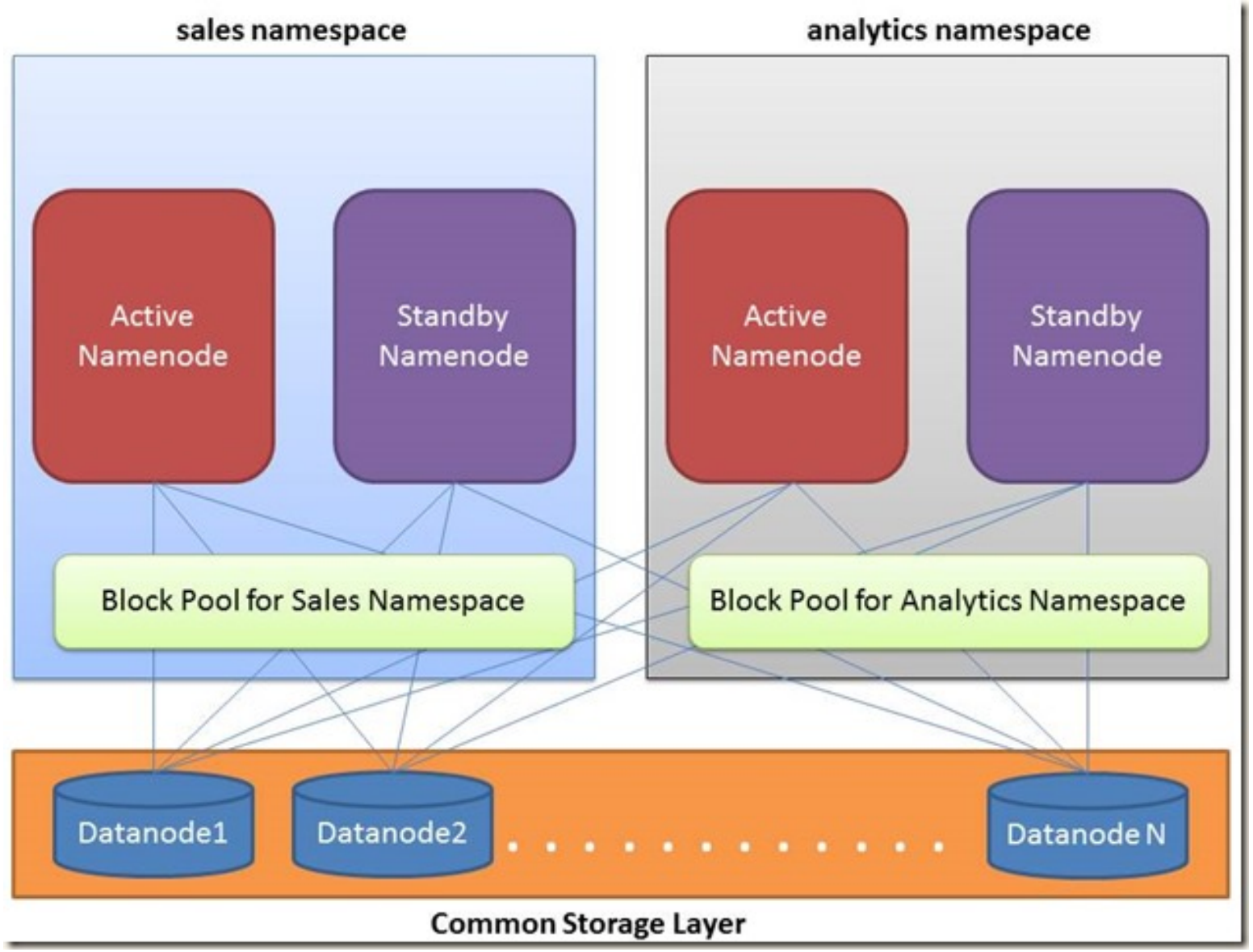
- Bottlenecks
- Chargeback
- Isolation

HDFS 2.0: High Availability Elaborated



Reference: Hadoop Summit 2012 | HDFS High Availability talk

HDFS 2.0: High Availability, Federated



http://pe-kay.blogspot.in/2016/02/configuring-federated-hdfs-cluster-with.html

Moving on to Parallel computing

Example: Find transactions with sale >= 10

- Sales by Product [say 32 inch LCD TV, ..] and Product category [say TV, Laptops…]

  [transaction level data]
  - Block 1 : product1 [key], 10[value]; product2, 15; product1, 5
  - Block 2 : product2, 40; product5, 15; product1, 55; product2, 10
  - Block 3 : product5, 30; product3, 25; product3, 15

- Product to product-category relation
  - Product 1 and 3 : PC1 ;
  - Product 2,4,5 : PC 2

Example: Create transactions by Product Category

- Sales by Product [say 32 inch LCD TV, ..] and Product category [say TV, Laptops…]

  [transaction level data]
  - Block 1 : product1 [key], 10[value]; product2, 15; product1, 5
  - Block 2 : product2, 40; product5, 15; product1, 55; product2, 10
  - Block 3 : product5, 30; product3, 25; product3, 15

- product to product-category relation

Product Category	Products
PC1	P1, P3
PC2	P2, P4, P5

Example: Find total sales by Product Category

- Sales by Product [say 32 inch LCD TV, ..] and Product category [say TV, Laptops…]
  - [transaction level data]
  - Block 1 : product1 [key], 10[value]; product2, 15; product1, 5
  - Block 2 : product2, 40; product5, 15; product1, 55; product2, 10
  - Block 3 : product5, 30; product3, 25; product3, 15
- product to product-category relation

Product Category	Products
PC1	P1, P3
PC2	P2, P4, P5

Let’s map what we did to MapReduce

- Map Tasks ➔
  - Mapper task 1 : product1 [key], 10[sale value]; product2, 15; product1, 5
  - Output: PC1, 10; *PC2, 15; PC1, 5*

  - Mapper task 2 : product2, 40; product5, 15; product1, 55; product2, 10
  - *Output: PC2, 40*; PC2, 15; PC1, 55; PC2, 10

  - Mapper task 3 : product5, 30; product3, 25; product3, 15
  - Output: PC2, 30; PC1, 25; PC1, 15

- Partitions [reducers] ➔by product category

Product Category	Products
PC1	P1, P3
PC2	P2, P4, P5

Shuffle, Sort and Partition

Product 1 and 3 : PC1 ; Product 2,4,5 : PC 2

- Data from Mappers:
- PC1, 10; *PC2, 15; PC1, 5*
- *PC2, 40*; PC2, 15; PC1, 55; PC2, 10
- PC2, 30; PC1, 25; PC1, 15

- PC1, 10
- PC1, 5
- PC1, 55 — Partition [reducer] 1 ➔ PC1, 110
- PC1, 25
- PC1, 15
- ----------
- **PC2, 15**
- **PC2, 40**
- **PC2, 15**
- **PC2, 10** — **Partition [reducer] 2➔ ???**
- **PC2, 30**;

MapReduce : Framework for parallelism

- MapReduce Job is a work that the client wants to be performed
- Is an execution of a **Mapper** and **Reducer** **tasks**.

Two processing layers/stages

- mapper and
- reducer.

A task in MapReduce is an execution of a Mapper or a Reducer on a slice of data

Though 1 block is present at 3 different locations by default, but framework allows only 1 mapper to process 1 block.

- So where should execution of mapper happen ?
- And how many map tasks ?

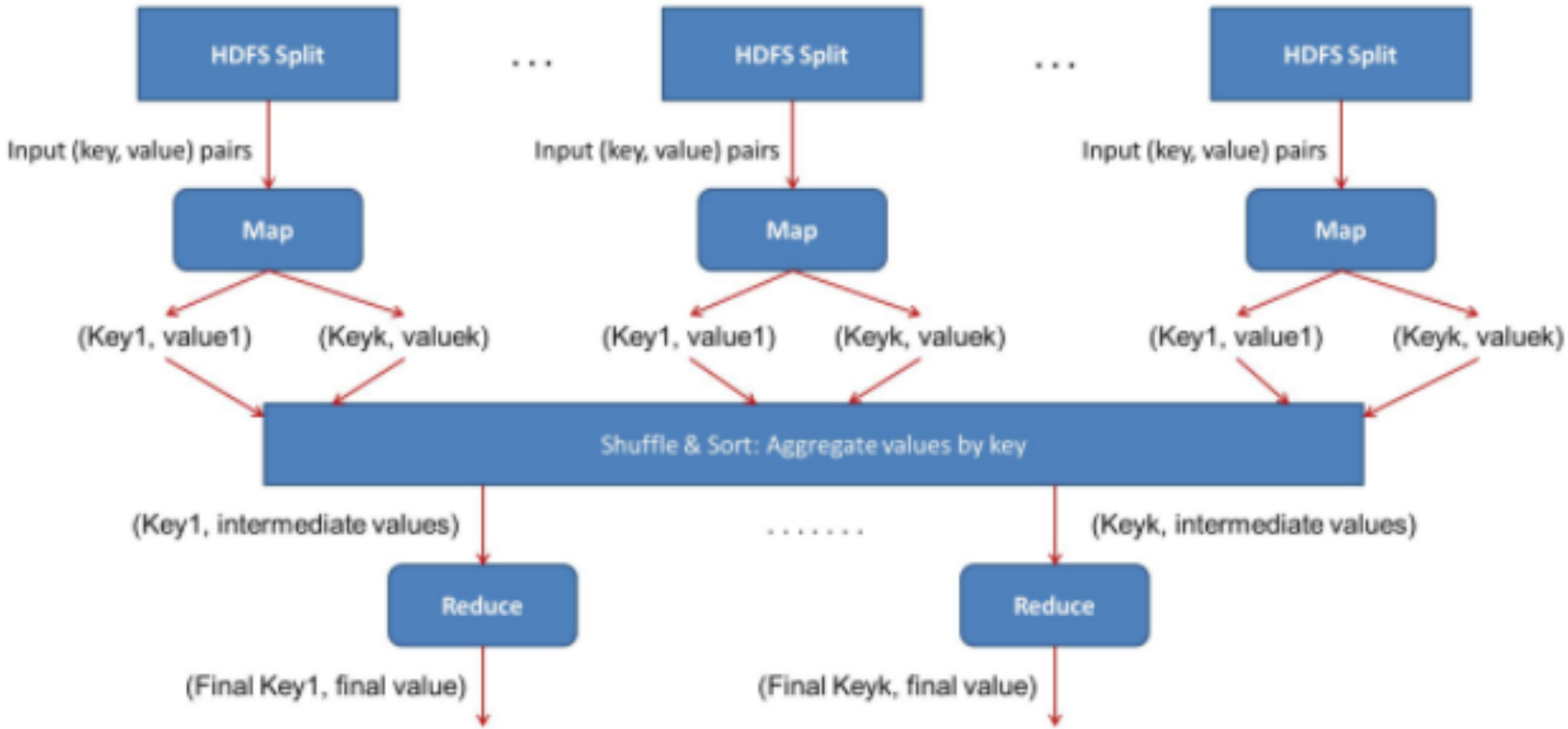
“where to execute” : Data Locality

- *Move computation close to the data rather than data to computation*”.

- A computation requested by an application is much more efficient if it is executed near the data it operates on when the size of the data is very huge.

- Minimizes network congestion and increases the throughput of the system

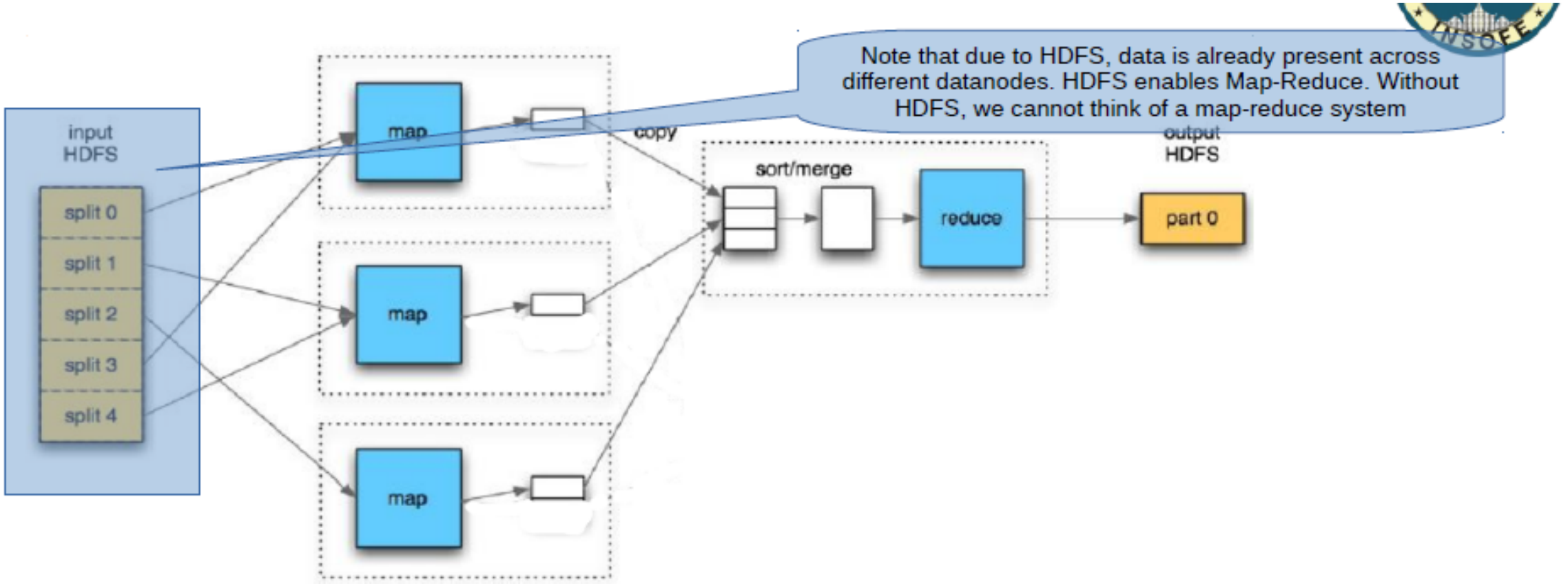
- Hadoop will try to execute the mapper on the nodes where the block resides.
  - In case the nodes [think of replicas] are not available, Hadoop will try to pick a node that is closest to the node that hosts the data block.
  - It could pick another node in the same rack, for example.


HDFS Split
HDFS Split
HDFS Split
Input (key, value) pairs
Input (key, value) pairs
Input (key, value) pairs
Map
Map
Map
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
(Key1, value1)
(Keyk, valuek)
Shuffle & Sort: Aggregate values by key
(Key1, intermediate values)
(Keyk, intermediate values)
Reduce
Reduce
(Final Key1, final value)
(Final Keyk, final value)

- **Mapper** writes the output to the local disk of the machine it is working.
- This is the temporary data. Also called intermediate output.

- As mapper finishes, data (output of the mapper) travels from mapper node to reducer node. Hence, this movement of output from mapper node to reducer node is called **shuffle**.

- An output from mapper is partitioned into many partitions;
  - Each of this partition goes to a reducer based on some conditions



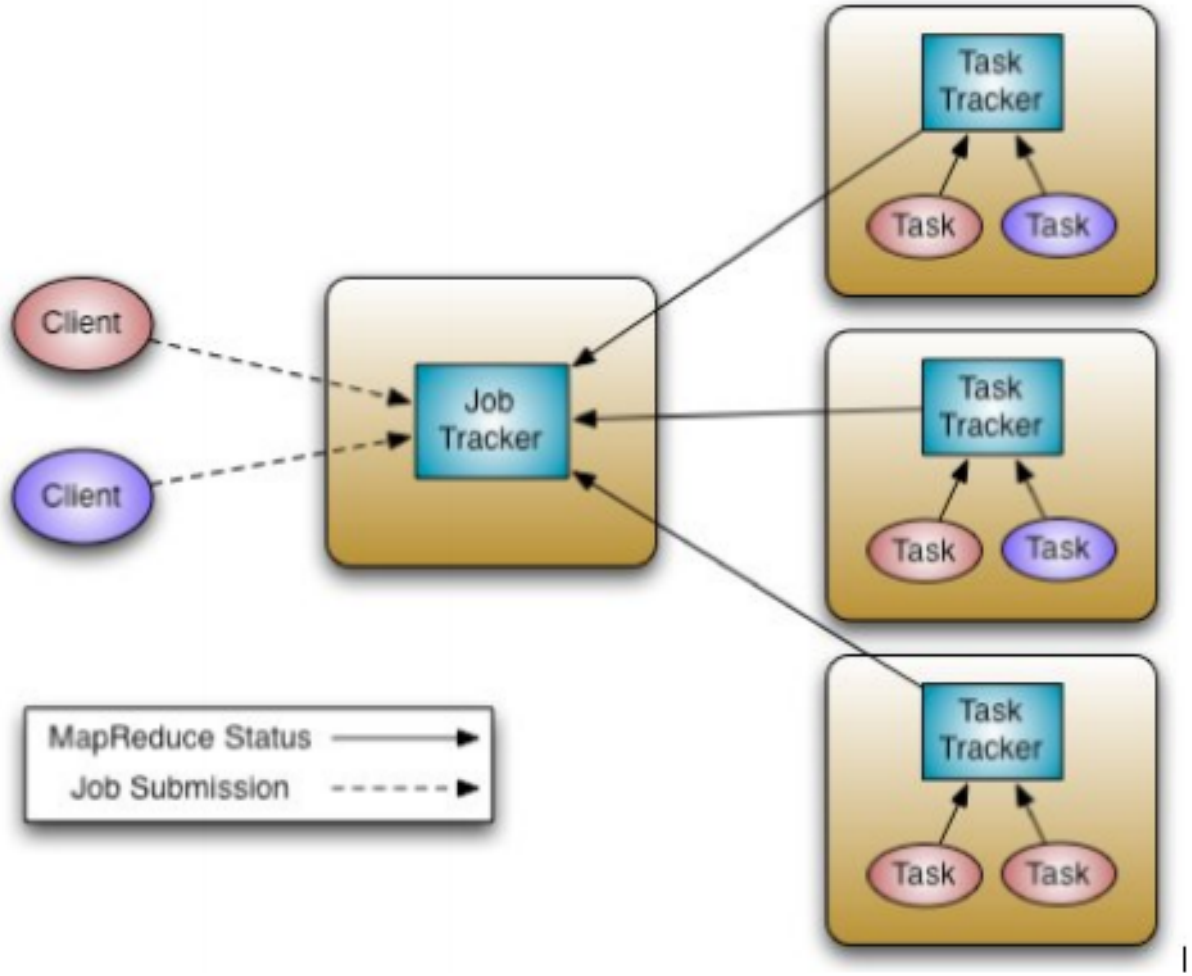
Ex: word counting of 300K documents spread across 3 data nodes

- Each data node:
  - Has ~100K documents, one mapper
  - Counts the word in its local set of 100K documents
- Reducer combines the results

- Let us say mapper 1 generates following counts by analyzing its files:
  - [Bombay, 1] [Alaska, 1] [Buffalo, 1][Bombay, 1] [Bombay, 1][Alaska, 1] [London, 1] [Zaire, 1] etc
  - It will send these results to the reducer

- Reducer gets results from all mappers to compute the final results
- Output will be written to HDFS
  - [Alaska, 2] [Bombay, 3] [Buffalo, 1][London, 1] [Zaire, 1]

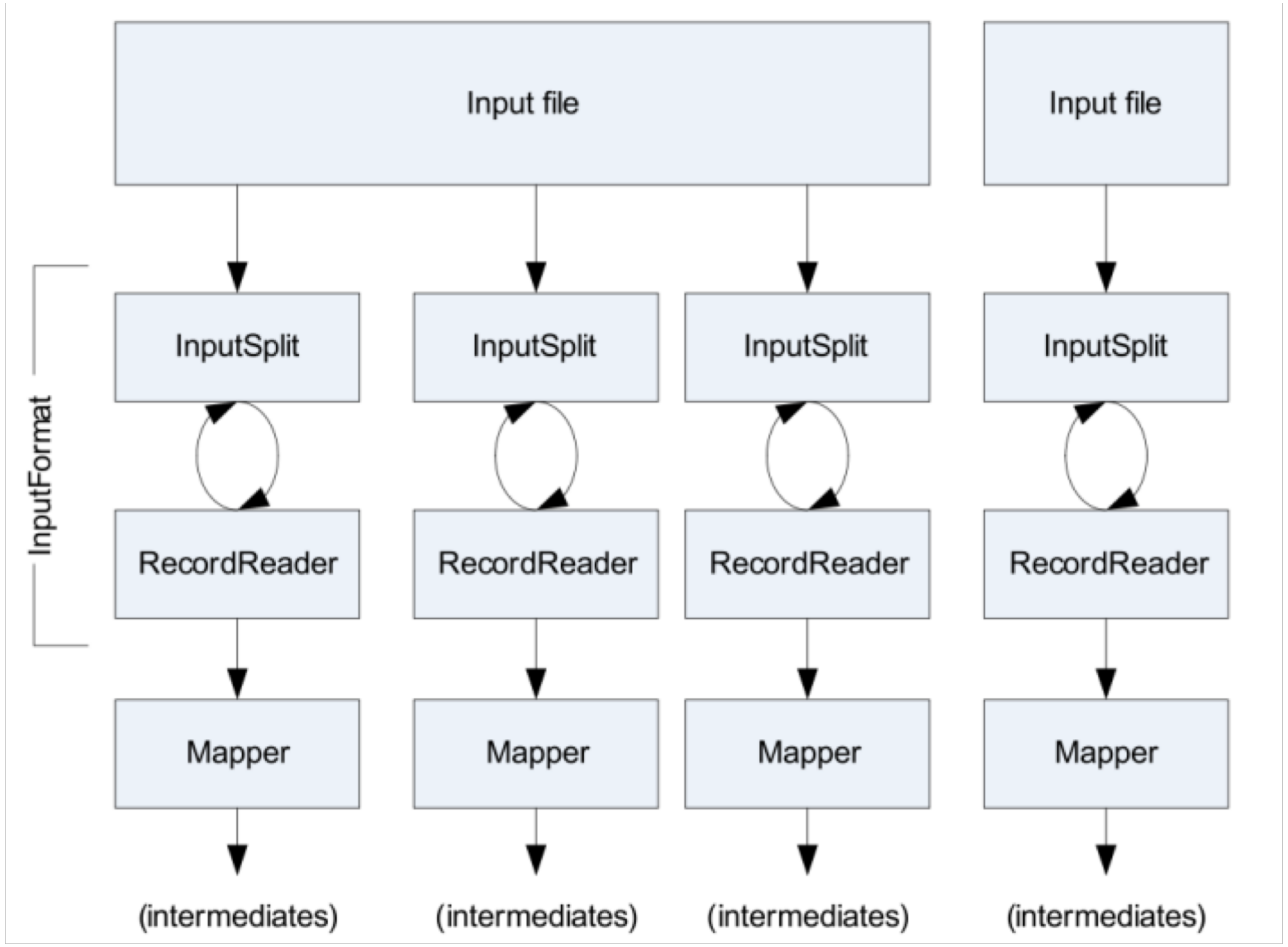
Hadoop 1 way of working



Internals of Map Reduce

FileInpoutFormat

DBInputFormat



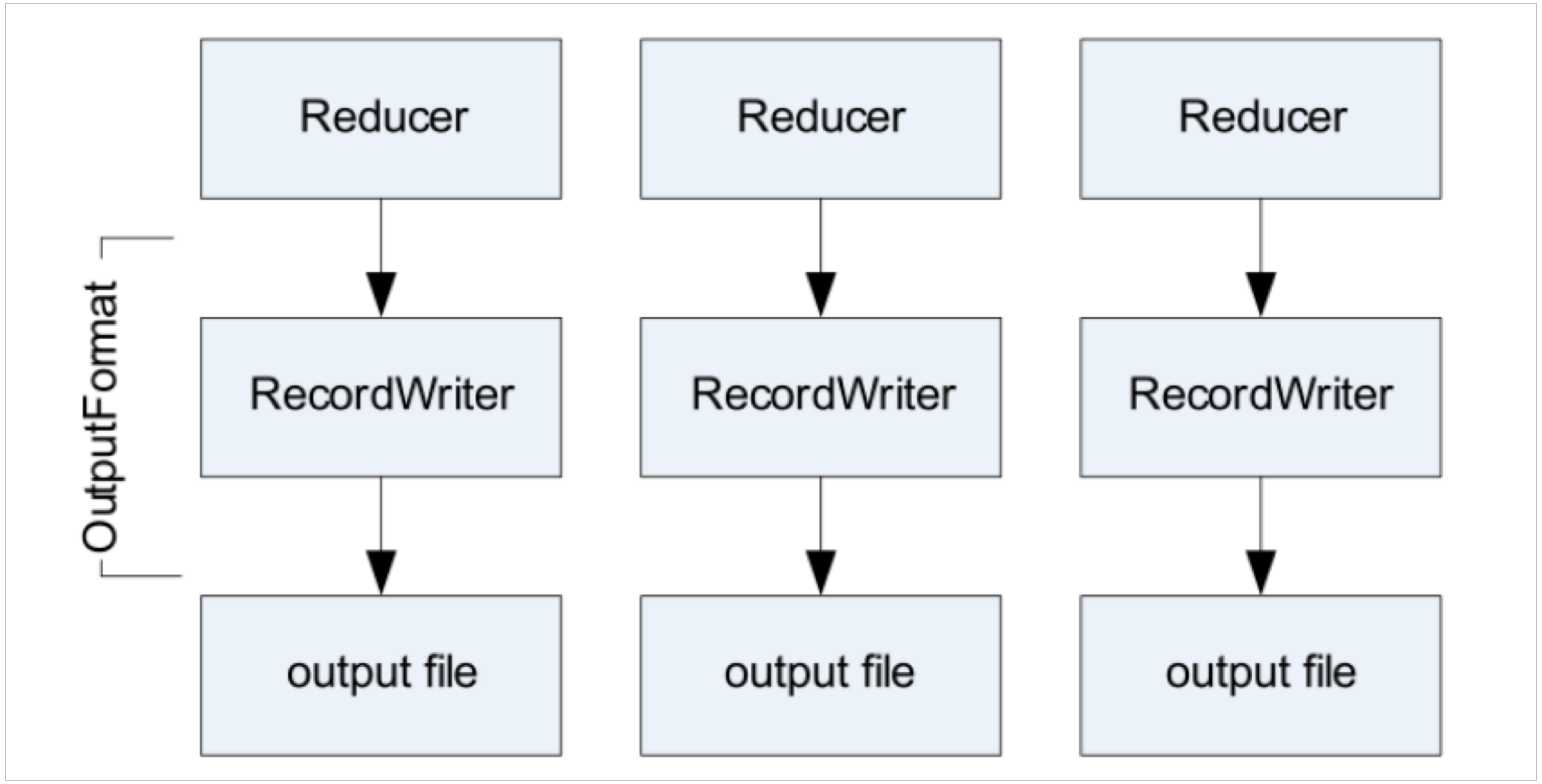
InputSplits are created by InputFormat

RecordReader's responsibility is to keep reading/converting data into key-value pairs until the end; which is sent to the mapper.

Number of map tasks will be equal to the number of InputSplits

Intermediate output is written to local disks

Same with Output Formats and Record Writers



Some features of MR jobs

- MapReduce jobs tend to be relatively short in terms of lines of code
- It is typical to combine multiple small MapReduce jobs together in a single workflow
  - Oozie
- You are likely to find that many of your MapReduce jobs use very similar [structurally] code. The logic in map and reduce will change.

How many mapper tasks?

Number of mappers set to run are completely dependent on :

1) File Size and

2) Block [split] Size

Main issues with MR1

- Can’t share resources with non MR applications
- Scalability of Job Tracker beyond 4000 nodes and 40,000 tasks is a problem
- In MR1, each node was configured with a fixed number of map slots and a fixed number of reduce slots…leading to sub optimal utilization

The Resource Management Layer

YARN

What is YARN

- Yet Another Resource Negotiator
- Replaces Job Tracker and Task Tracker architecture in MR1
- YARN designed to scale up to 10,000 nodes and 100,000 tasks.
- Under YARN, there is no distinction between resources available for maps and resources available for reduces – all resources are available for both; the notion of slots has been discarded
- Resources are now configured/allocated in terms of amounts of memory (in megabytes) and CPU

YARN

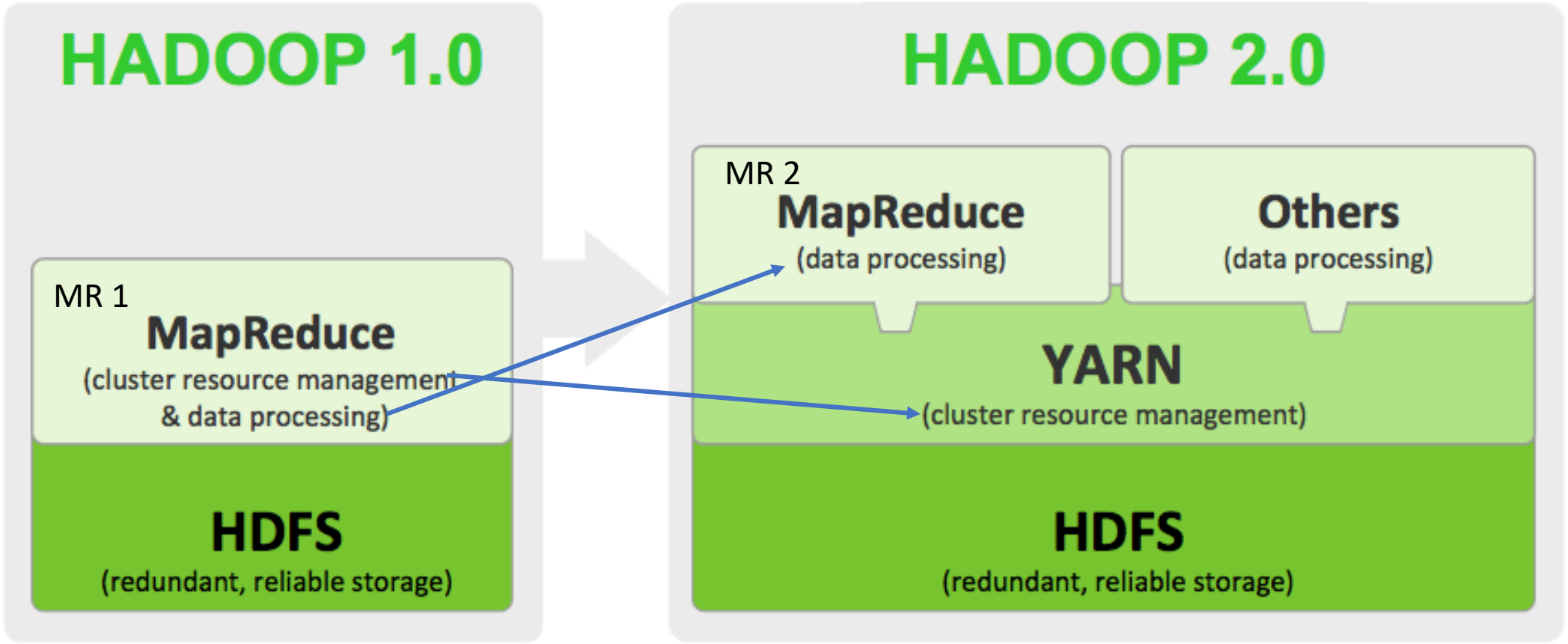
- YARN splits up the two major responsibilities of the JobTracker:
  - Resource management and
  - Job scheduling/monitoring

- Two separate daemons:

  [1] a global ResourceManager and

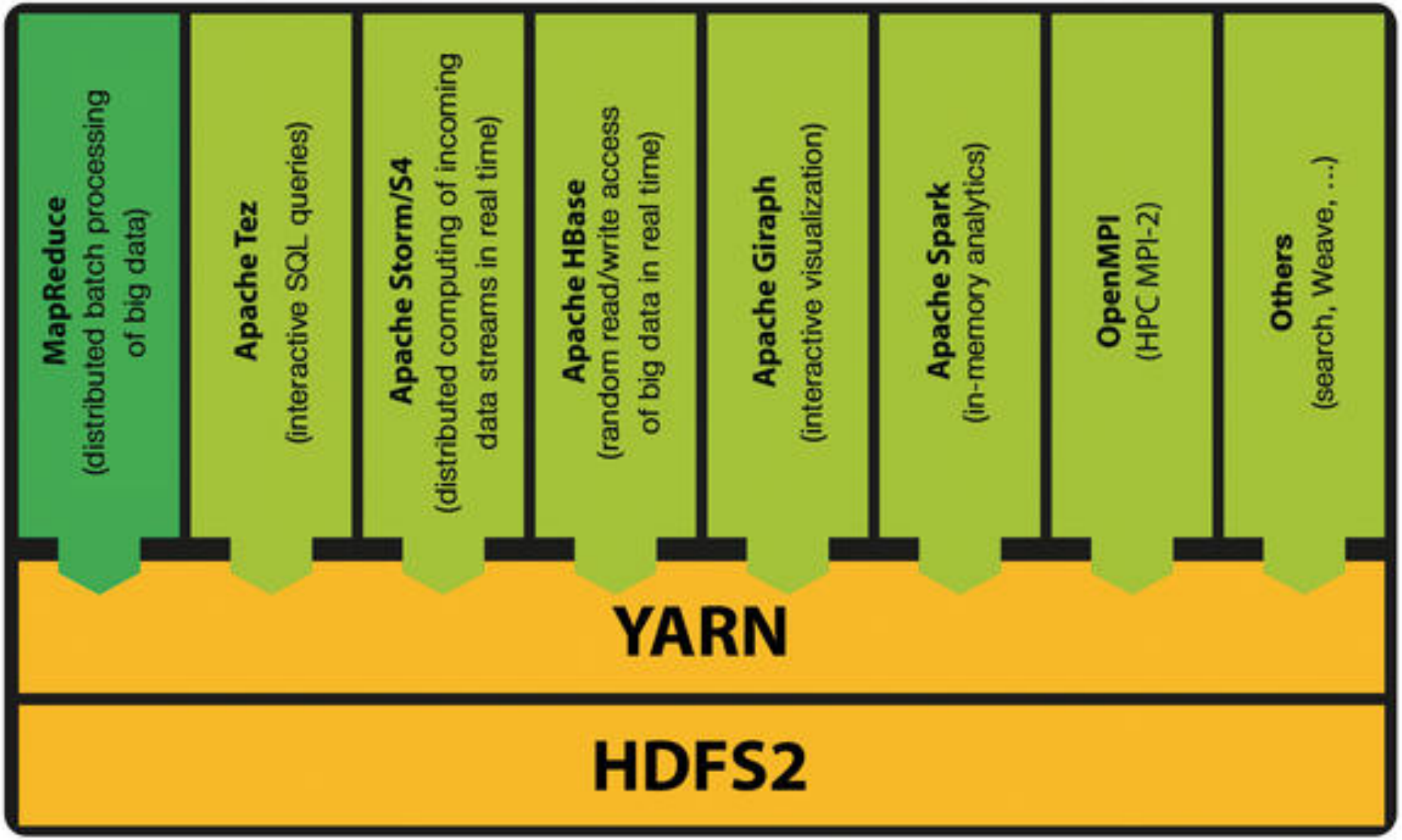
  [2] per-application ApplicationMaster (AM)

- Enables non-MapReduce tasks to work within a Hadoop installation

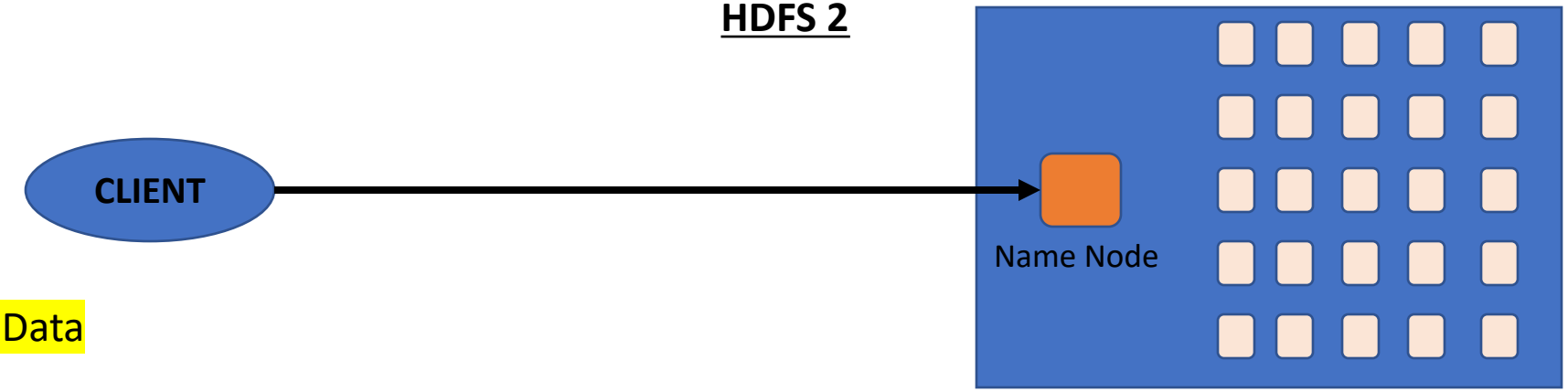


Provides full backward compatibility with existing MapReduce tasks and applications
Hadoop 2 includes a MR Application Master to manage MR Jobs

YARN powers many Hadoop layers



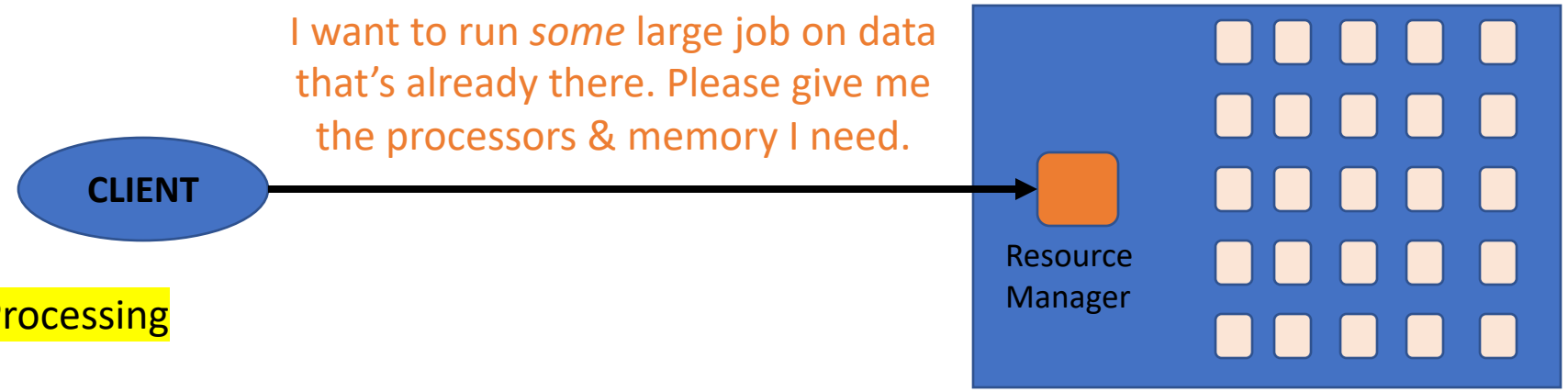
HDFS 2



Distributed Data

YARN

I want to run *some* large job on data that's already there. Please give me the processors & memory I need.



Distributed Processing

Roles of various components:

Resource Manager:

- acts as the sole arbitrator of cluster resources.
- ultimate authority that arbitrates resources among all the applications.
- responsible for optimizing cluster utilization

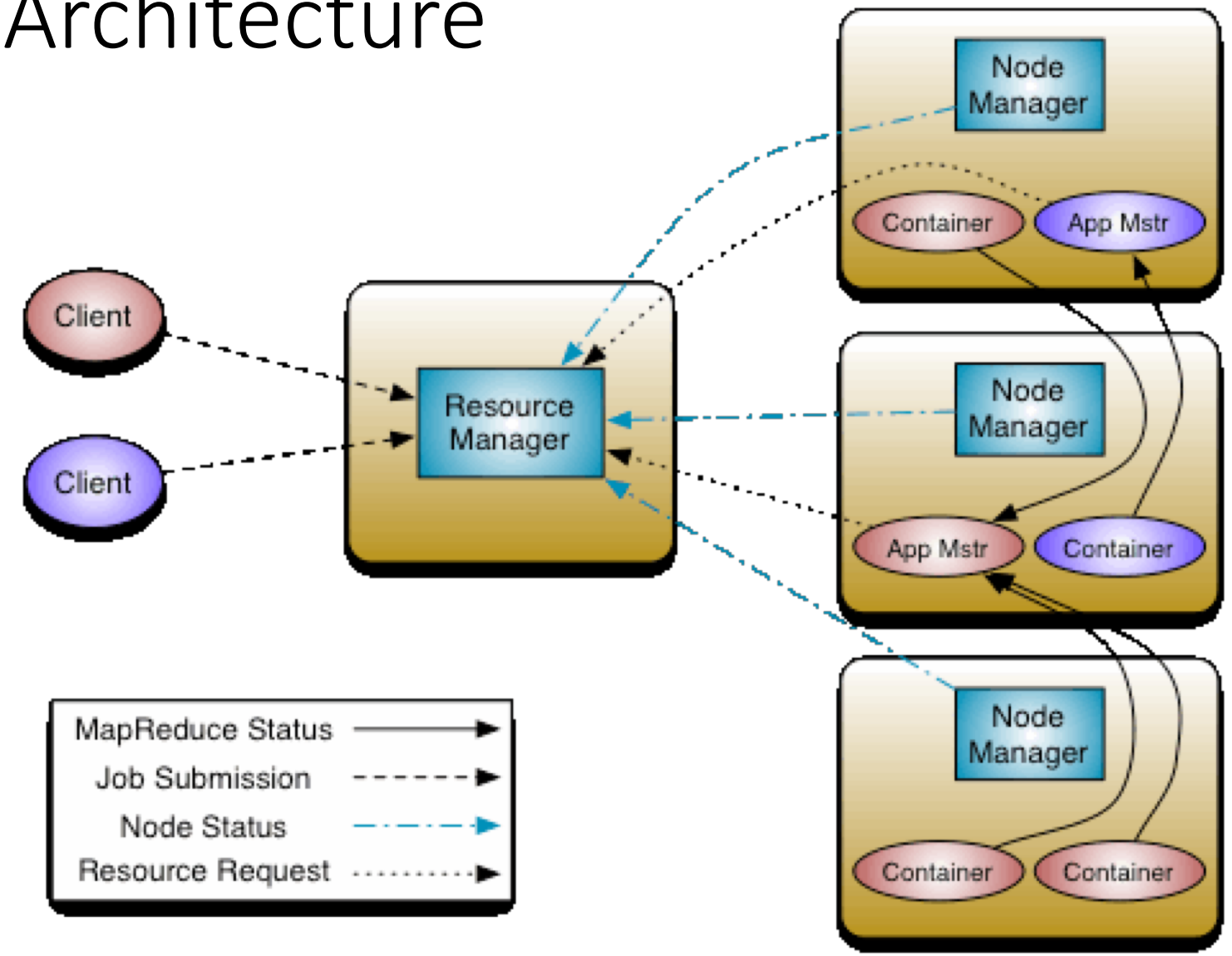
User applications, including MapReduce jobs, ask for specific resource requests via the ApplicationMaster component, which in in-turn negotiates with the Resource Manager to create an Application Container within the cluster.

New Concepts:

[1] Application Master [per application] Example: MRAppMaster

[2] Application Container

YARN Architecture



The NodeManager:

- per-machine slave [daemon]
- responsible for launching the applications' containers,
- monitoring their resource usage (CPU, memory, disk, network)
- Reports to the ResourceManager.

The YARN system (ResourceManager and NodeManager) has to protect itself from faulty or malicious ApplicationMaster(s) and resources granted to them at all costs.

The per-application ApplicationMaster : has the responsibility of negotiating appropriate resource containers with RM, tracking container status and monitoring progress

AM also works with the NodeManager(s) to execute and monitor the containers and their resource consumption

What is container and How to get container

- Container is the resource allocation [container ID, Node Manager], which is the successful result of the ResourceManager granting a specific ResourceRequest from AM

- RM responds to a resource request by granting a container, which satisfies the requirements laid out by the ApplicationMaster in the initial ResourceRequest.

- A ResourceRequest format:

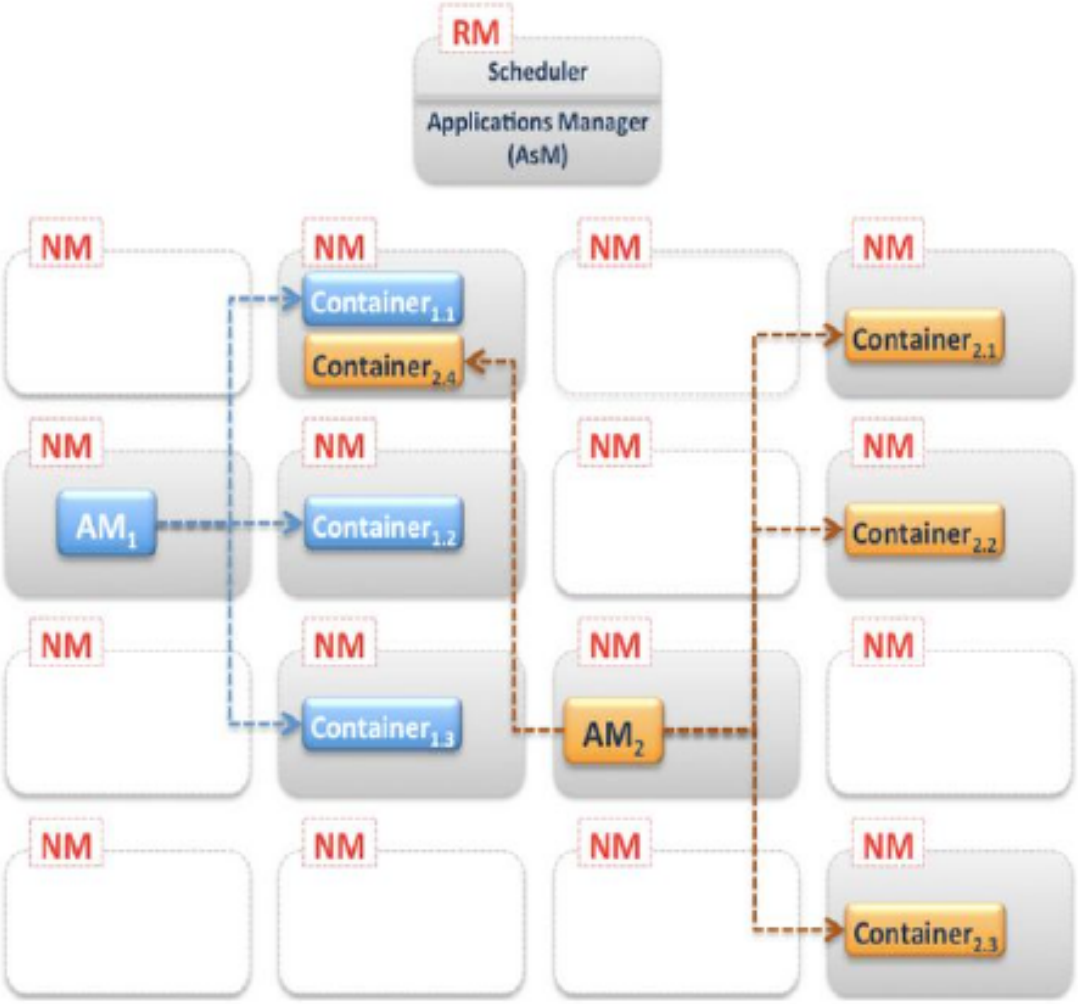
  <resource-name, priority, resource-requirement, number-of-containers>

  - Resource requirements: CPU and Memory

- ApplicationMaster has to take the Container and present it to the NodeManager managing the host, on which the container was allocated, to use the resources for launching its tasks.

Sequence of actions:

- Client program submits the app
- ResourceManager negotiates a container to start the ApplicationMaster
  - When container is available, it launches the ApplicationMaster
- ApplicationMaster on boot-up registers with the ResourceManager
- ApplicationMaster negotiates appropriate resource containers
  - On successful container allocations, ApplicationMaster launches the container by providing the container launch specification (with app code) to NodeManager
  - App code executing within the container provides necessary information (progress, status etc.) to its ApplicationMaster via an application-specific protocol.
- During the app execution, client that submitted the app communicates directly with the ApplicationMaster to get status, progress updates etc.
- Once app is complete, and all necessary work has been finished, the ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed



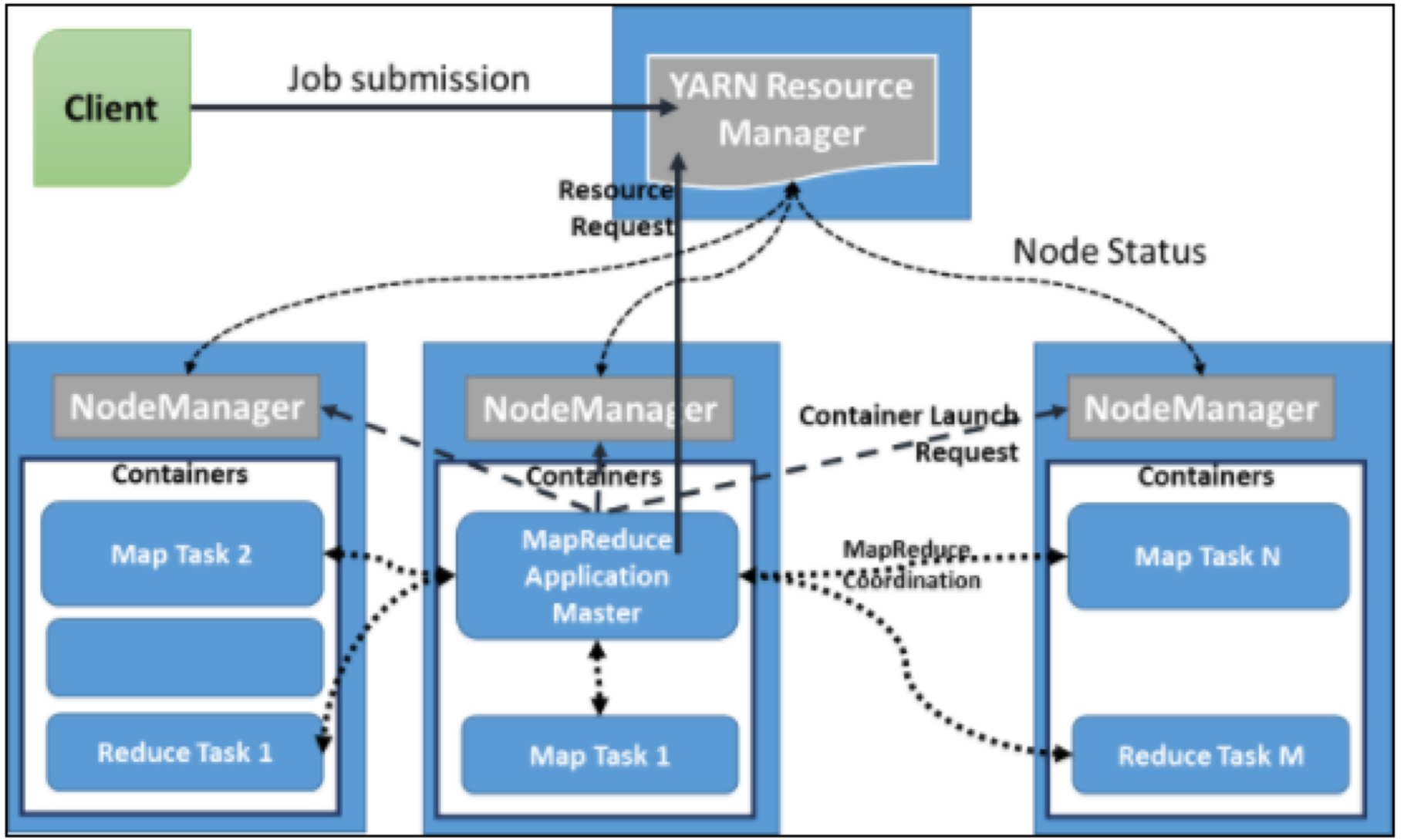
Example: Application 1 (Blue) is using three containers, and Applicaton 2 (Brown) is using four containers.

Launching the container

- The ApplicationMaster has to provide more details to the NodeManager to actually launch the container.

- The Container launch specification API is platform agnostic and contains:
  - Command line to launch the process within the container.
  - Environment variables
  - Local resources necessary on the machine prior to launch, such as jars, shared-objects, auxiliary data files etc

MR [v2] On YARN


Client
Job submission
YARN Resource Manager
Resource Request
Node Status
NodeManager
NodeManager
NodeManager
Container Launch Request
Containers
Containers
Containers
Map Task 2
MapReduce Application Master
Map Task N
MapReduce Coordination
Reduce Task 1
Map Task 1
Reduce Task M

Role of a Resource Manager

- Manages nodes
  - Tracks heartbeats from NodeManagers
- Manages containers
  - Handles AM requests for resources
  - De-allocates containers when they expire or the appl completes
- Manages ApplicationMasters
  - Creates a container for AMs and tracks heartbeats
- Manages security

Role of a Node Manager

- **What it does**
  - Communicates with the RM
    - Registers and provides info on node resources
    - Sends heartbeats and container status
  - Manages processes in containers
    - Launches AMs on request from the RM
    - Launches application processes on request from AM
    - Monitors resource usage by containers; kills run-away processes
  - Provides logging services to applications
    - Aggregates logs for an application and saves them to HDFS
  - Runs auxiliary services
  - Maintains node level security via ACLs

So what all we have learnt ?

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering