Chapter 3 : Yarn

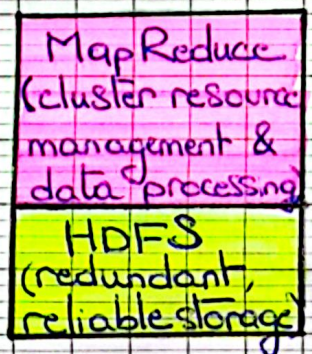
→ What is YARN ?

- Stand for "yet another recource negotiator"
- It's a resource managment platform that was introduced in Hadoop 2.0
- Overcome the limitations of MapReduce framework

- Key purpose: Yarn separates two critical functions that were previously combined
  - ↳ Resource Managment : Managing cluster resources (CPU, memory, etc.)
  - ↳ Processing Engine : Running applications (MapReduce, Spark, etc.)
- This separation allows multiple processing Fromeworks to run on the same Hadoop cluster simultaneously

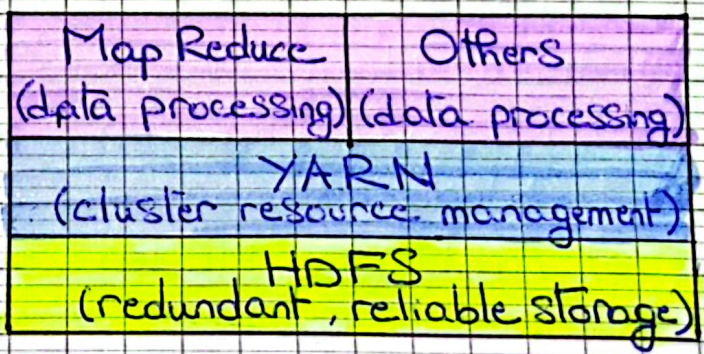
Hadoop V1

Users could only write MapReduce programs

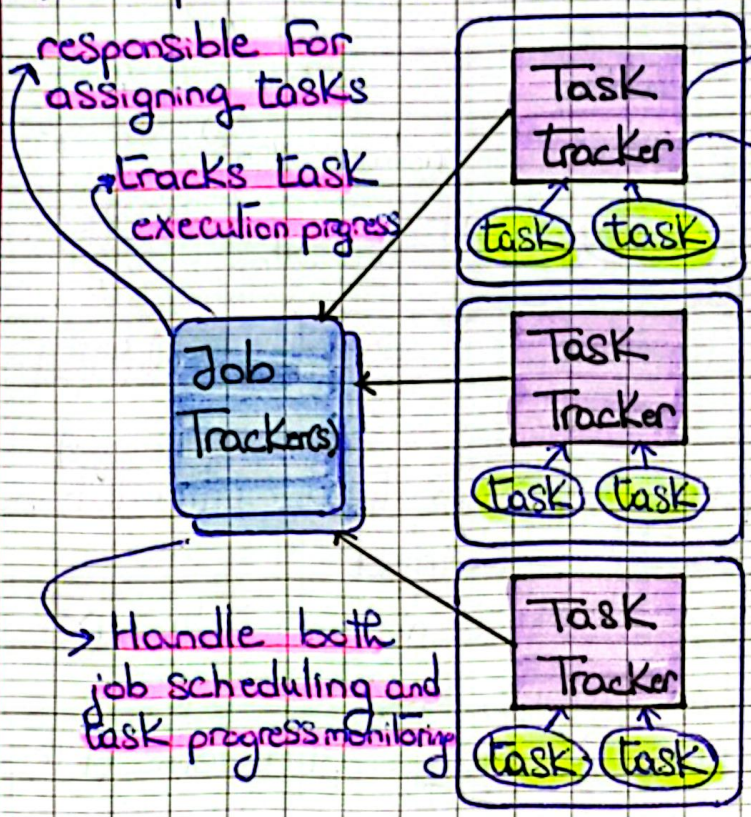


Hadoop V2

Users could work on multiple processing models in addition to MapReduce (like Spark)



→ MapReduce v1 and its Limitations



↳ Limitations

- Scalability limitation (4000 nodes, 40000 tasks)
- Computational resources divided into fixed nb. of map and reduce slots
- designed exclusively for MapReduce (non-mapreduce applications cannot run)

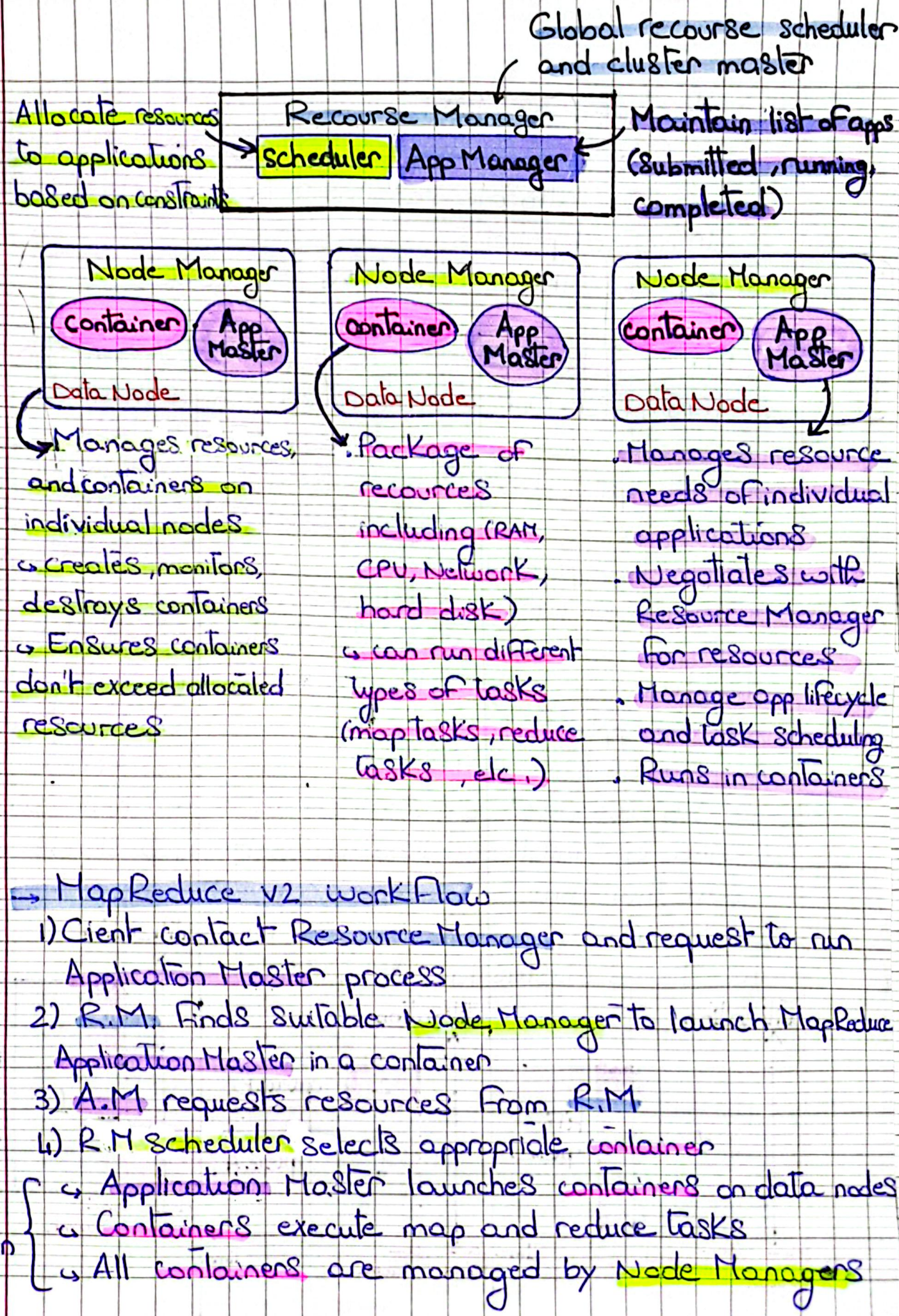
↳ Yarn overcome these limitations:

- scale up to 10 000 nodes and 100 000 tasks
- allocates dynamic containers
- resource manager will not be overloaded so the cluster and the tasks will be more available

→ YARN Architecture Overview

- Three Main Components
  1) Resource Manager (The master)
  2) Node Manager (The workers)
  3) Application Master (Per-application coordinator)

- Additional Key Concepts
  4) Container (resource allocation unit)
  5) Client (submits applications)



→ MapReduce V2 workFlow

1) Client contact Resource Manager and request to run Application Master process
2) R.M. finds suitable Node Manager to launch MapReduce Application Master in a container
3) A.M requests resources from R.M
4) R M scheduler selects appropriate container
5) task execution
   - ↳ Application Master launches containers on data nodes
   - ↳ Containers execute map and reduce tasks
   - ↳ All containers are managed by Node Managers

→ Scheduling in YARN

Scheduler in yarn is part of Resource Manager

↳ purpose: allocate resources to application according to defined policies

↳ Types : Three schedulers available

A. FIFO Scheduler

. First-In, First-Out

. Characteristics:

↳ Simple queue-based approach

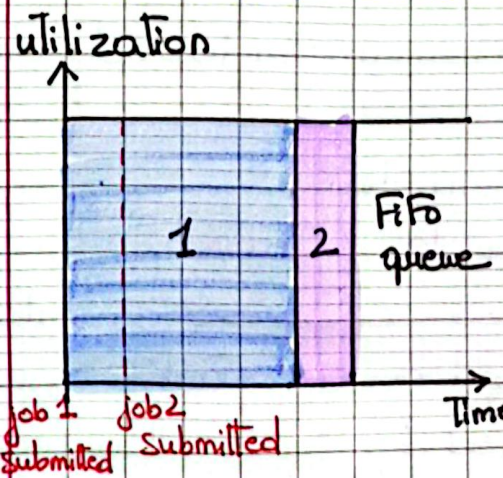
↳ Applications processed in submission order

. Problems:

↳ Large applications consume all cluster resources

↳ other apps must wait their turn

↳ not suitable for shared clusters



B. Capacity Scheduler

. Queue based with capacity reservation

. Characteristics

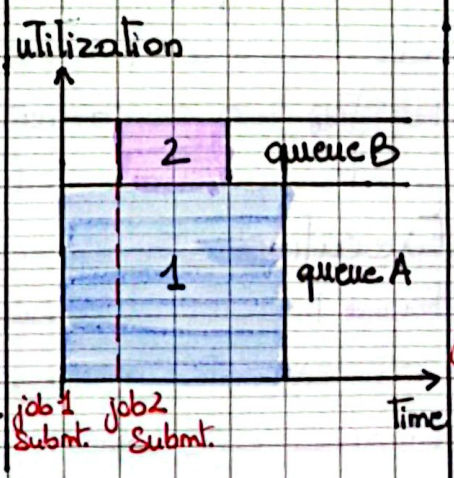
↳ Separate dedicated queues

↳ Small jobs can start immediatly in their queue

. Problems:

↳ overall cluster utilization cost (reserved capacity)

↳ Large jobs finish later than FIFO



C. Fair Scheduler

. Dynamic resource balancing

. Characteristics

↳ no capacity reservation needed

↳ resources dynamically balanced between running jobs

↳ First job gets all resources initially

↳ when second job starts, resources are split Fairly