CSP554
BIG DATA TECHNOLOGIES

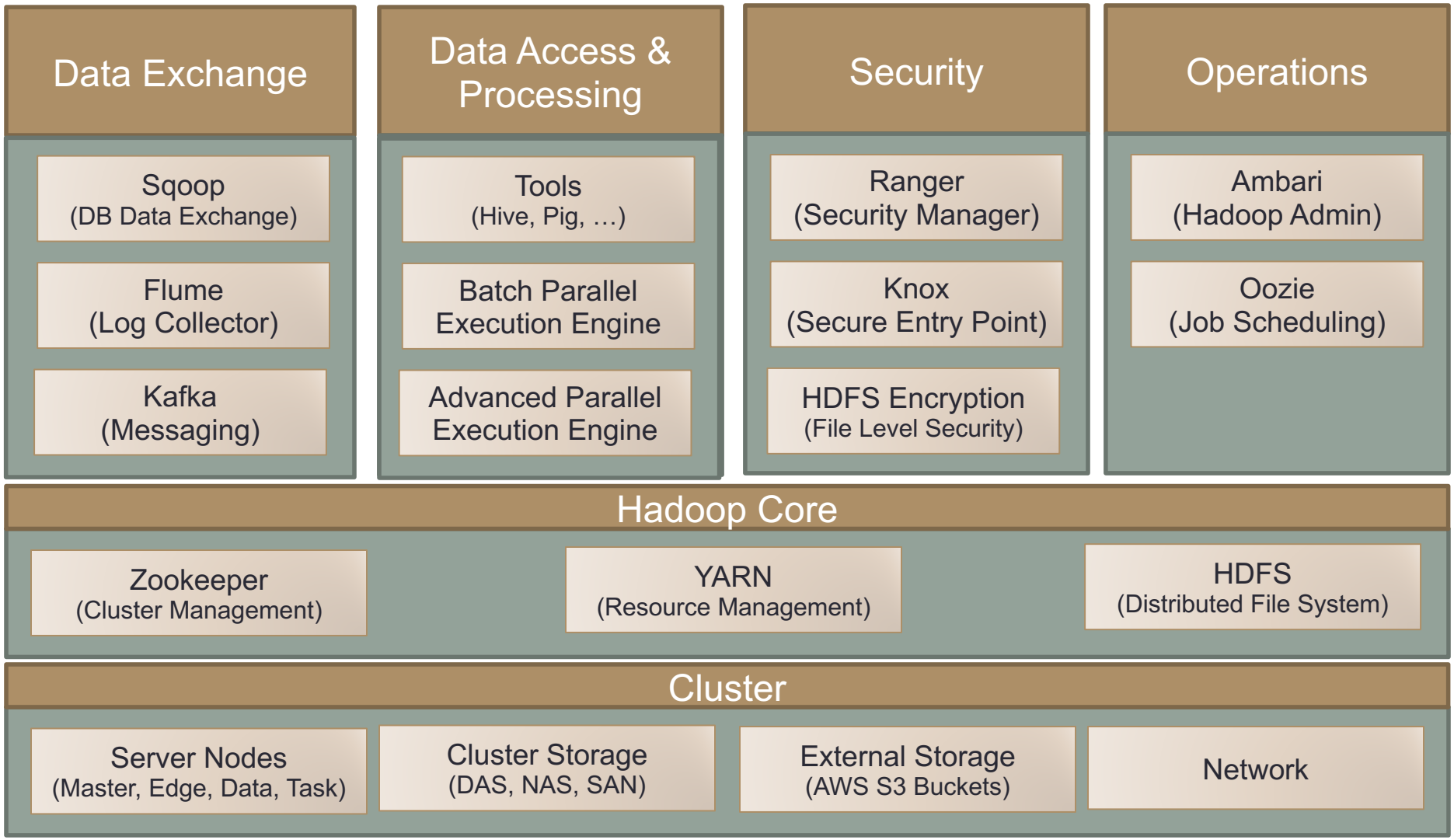
Module 3a

YARN

Core Hadoop

- Hadoop Common
  - The common utilities that provide basic support to other Hadoop modules.
- Hadoop Distributed File System
  - Enables storage of large amounts of data in redundancy over a cluster of commodity machines,
- Zookeeper
  - A centralized service for maintaining Hadoop cluster configuration information, naming and providing distributed synchronization
- Hadoop YARN:
  - A framework that takes care of cluster resource management and job scheduling tasks

Hadoop Architecture Landscape



YARN (Yet Another Resource Manager)

- Hadoop's cluster resource management and application scheduling system
- Provides a service and framework independent means to control cluster utilization and manage applications
- Supports multitenancy where cluster resources are shared among multiple users according to well defined policies
- Facilitates higher cluster utilization, whereby resources not used by one user could be made available to another
- Allows lower operational costs as only one cluster may be provisioned, secured, managed and tuned
  - While supporting a range of services and variable workloads
- Reduces data motion; no need to move data between cluster running one Hadoop service and cluster running another

YARN (Yet Another Resource Manager)

- Manageable resources include CPU and memory
- No support (yet) for resources such as GPU, disk, network
- Applications can request resources conforming to constraints on node or rack locality
  - If a locality based request can't be satisfied in a timely manner resources are allocated from the next node to heartbeat

YARN (Yet Another Resource Manager)

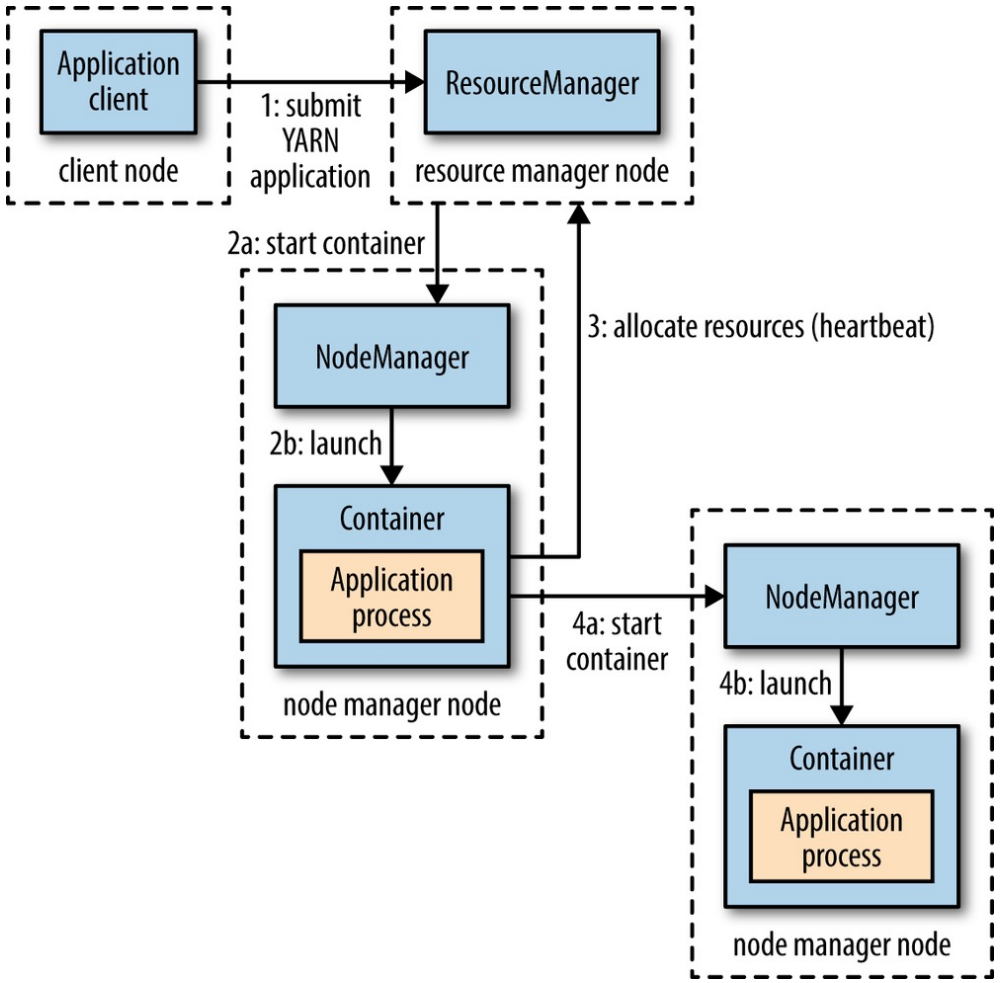
- A *cluster* is made up of two or more hosts connected by an internal high-speed network
  - *Master hosts* are a small number of hosts reserved to control the rest of the cluster. *Worker hosts* are the non-master hosts in the cluster.
- In a cluster with YARN running…
  - The master process is called the *ResourceManager* and the worker processes are called *NodeManagers*
- YARN keeps track of two *resources* on the cluster….
  - *vcores* and *memory*
- The NodeManager on each host keeps track of the local host's resources…
- And the ResourceManager keeps track of the cluster's total

YARN (Yet Another Resource Manager)

- A *container* in YARN holds resources on the cluster
- YARN determines where there is room on a host in the cluster for the size of the hold for the container
- Once the container is allocated, those resources are usable by the container
- An *application* in YARN comprises three parts:
  - The *application client*, which is how a program is run on the cluster
  - An *ApplicationMaster* which provides YARN with the ability to perform allocation on behalf of the application
  - One or more *tasks* that do the actual work (runs in a process) in the container allocated by YARN
- A *MapReduce application* consists of *map tasks* and *reduce tasks*
- A MapReduce application running in a YARN cluster runs with the addition of an ApplicationMaster as a YARN requirement

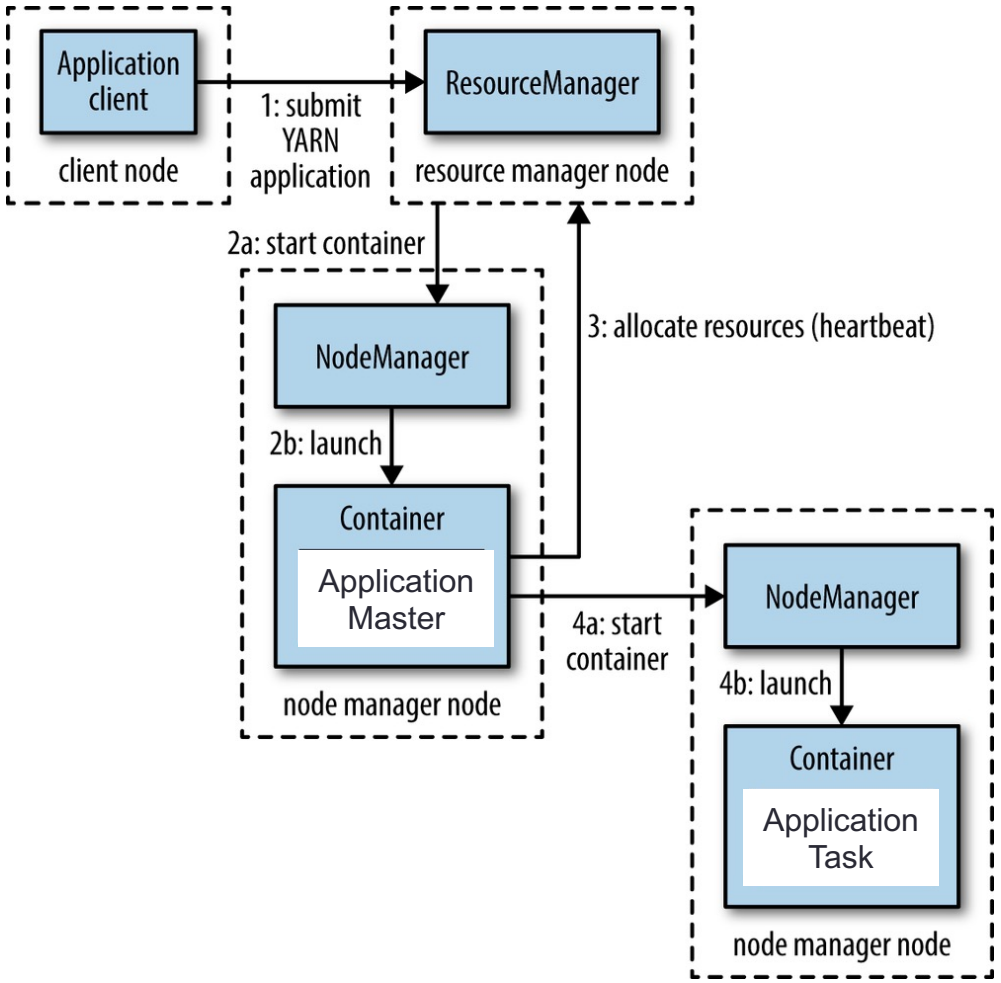
YARN Architecture

- YARN provides its core services via two types of long-running daemon
  - a *resource manager* (one per cluster) to manage the use of resources across the cluster
  - and *nodemanagers* running on all the nodes in the cluster to launch and monitor *containers*
    - A container executes an application-specific process with a constrained set of resources (memory, CPU, and so on)
    - Depending on how YARN is configured a container may be a Unix process or a Linux cgroup



YARN Architecture

- A client contacts the resource manager and asks it to run an *application master* process
- The resource manager then finds a node manager that can launch the application master in a container
- What the application master does once it is running depends on the application
  - It could run a computation in the container it is running in and return the result to the client
  - Or it could request more containers from the resource managers and use them to run a distributed computation
  - This is what the MapReduce YARN application does



YARN Architecture

- In the common case of launching a container to process an HDFS block to run a map task in MapReduce the application will request a container on one of the nodes hosting the block's three replicas,
- A YARN application can make resource requests at any time while it is running
  - an application can make all of its requests up front
  - or it can take a more dynamic approach where it requests more resources dynamically to meet the changing needs of the application
- Spark takes the first approach, starting a fixed number of executors on the cluster
- MapReduce has two phases: the map task containers are requested up front, but the reduce task containers are not started until later.

Caution, Overloaded Terms Ahead

- YARN uses some common terms in uncommon ways
- When most people hear “container”, they think Docker
- In the Hadoop ecosystem, it takes on a new meaning
- A *Resource Container* represents a collection of physical resources such as RAM and CPU cores
- It is an abstraction used to bundle resources into distinct and trackable units
- In effect a container is a *right* to use a specified amount of resources

Caution, Overloaded Terms Ahead

- "Application" is another overloaded term
- Most people think of an application as a process which executes to provide some business capability
- In YARN, an *application* represents a set of one or more tasks that are to be executed to accomplish some overall objective
  - So tasks are the logical units (processes, threads) of work of applications
- An application in this context is also referred to as a type of job
- For example, a MapReduce job involves the execution of some number of map and reduce tasks

YARN Architecture
Resource Manager

- The Resource Manager has two main components
  - Scheduler
  - Applications Manager
- The Scheduler is responsible for allocating resources to the various running applications
- It performs no monitoring or tracking of status for the application
- It offers no guarantees about restarting failed tasks either due to application failure or hardware failures.
- The Scheduler has a pluggable resource allocation policy which is responsible for sharing cluster resources

YARN Architecture

Applications Manager

- The Applications Manager is responsible for accepting client requests to execute applications
- It negotiates the first container executing the application specific Application Master task
- It provides the service for restarting the Application Master task on failure
- Receives heartbeat indications from Node Managers to track Worker Node status and availability

YARN Architecture
Resource Scheduling Policies

- A cluster scheduler essentially has to address:
  - Multi-tenancy
    - Users launch many different applications, on behalf of multiple departments or organizations (all of which have some partial “ownership” of the cluster)
    - A cluster scheduler must share resources in some equitable fashion among these parties
  - Scalability
    - A cluster scheduler needs to scale to large clusters running many applications
    - This means that increasing the size of the cluster should improve performance without negatively affecting system latencies.
    - Also resource scheduling should not be optimized for any specific application type

YARN Architecture
Resource Scheduling Policies

- There are two standard modem resource scheduling policies
  - Fair Scheduler
    - Allocates resources to weighted pools, with fair sharing within each pool.
  - Capacity Scheduler
    - Allocates resources to pools, with FIFO scheduling within each pool
- Different vendors default to different scheduling policies
  - EMR => Capacity scheduler
  - Cloudera => Fair Scheduler

YARN Architecture

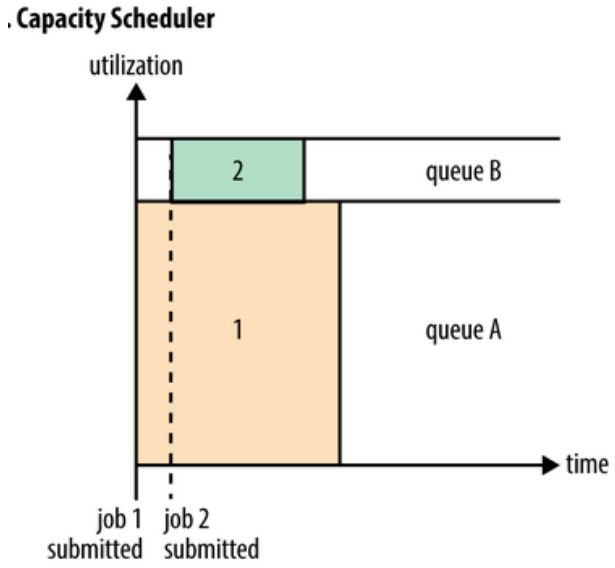
Resource Scheduling Policies

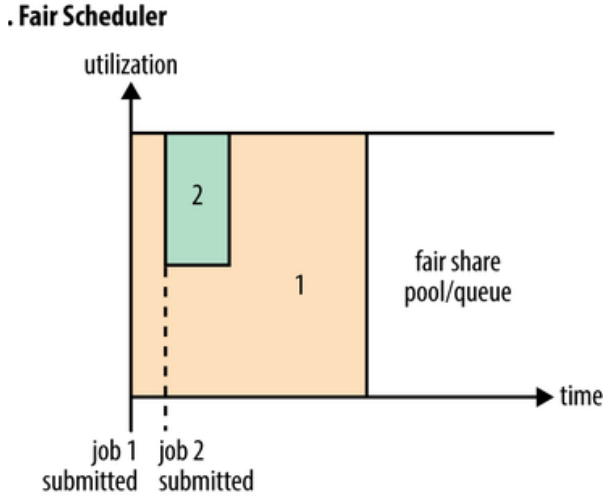
- The Fair Scheduler is more flexible and allows for jobs to consume unused resources in the cluster
  - It provides resource guarantees by preempting tasks above the guarantee (fair share) based on defined weights
  - This makes it a good default for small to medium sized clusters.
- The Capacity Scheduler is more rigid, and defines queues with resource quotas
  - Jobs cannot consume extra resources
  - This requires more configuration, tuning, and capacity planning
  - It's generally used on large clusters with lots of different workloads with different needs.

Scheduler Options

- With the Capacity Scheduler a separate dedicated queue allows the small job to start as soon as it is submitted, although this is at the cost of overall cluster utilization since the queue capacity is reserved for jobs in that queue
  - AWS EMR clusters by default are configured with a single capacity scheduler queue and can run a single job at any given time.

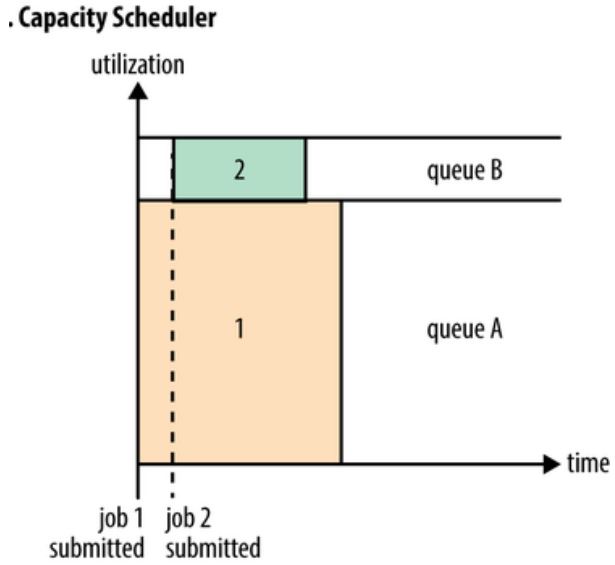
- With the Fair Scheduler there is no need to reserve a set amount of capacity, since it will dynamically balance resources between all running jobs
  - Just after the first (large) job starts, it is the only job running, so it gets all the resources in the cluster
  - When the second (small) job starts, it is allocated half of the cluster resources so that each job is using its fair share of resources





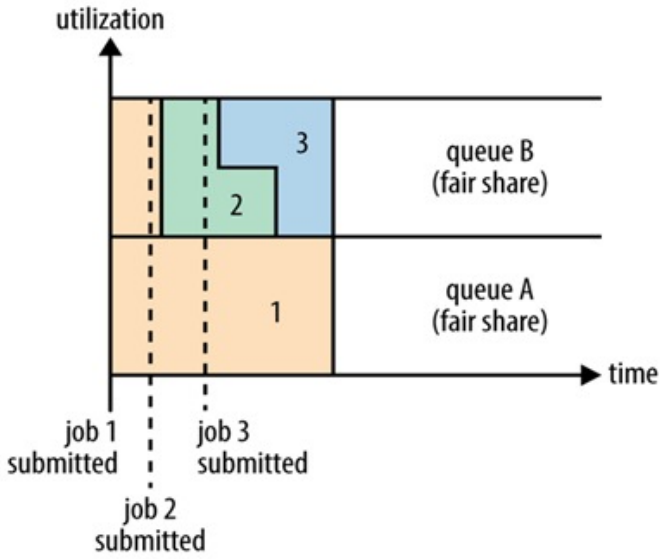
Scheduler Options

- The Capacity Scheduler is designed to allow sharing a large cluster while giving each department a minimum capacity guarantee
- The central idea is that the available resources in the Hadoop Map-Reduce cluster are partitioned among multiple departments who collectively fund the cluster based on computing needs
- There is an added benefit that an department can access any excess capacity not being used by others
- This provides elasticity for departments in a cost-effective manner



Scheduler Options

- Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time
- When there is a single job running, that job uses the entire cluster
- When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time.
- Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs



YARN Architecture

Node Manager

- Per worker node agent tasked with
  - Overseeing node containers through their lifecycles
  - Including launching containers (application tasks)
  - Monitoring node container resource usage
  - Periodically communicating node liveness to the Resource Manager

YARN Architecture

Application Master

- Neither the resource manager or the node manager controls overall progress of an application
- This makes sense as we want to keep our distributed middleware (Resource Manager and Node Manager) application agnostic
- So the question becomes, how is the progress of each application managed?
- This is the role of the application specific and custom Application Master task
- Each type of application running on Hadoop has its type of Application Master
- Each running instance of an application has its dedicated Application Master instance
- This instance lives in its own separate container on one of the nodes in the cluster
- It is the one container (task) whose initiation is performed by the Resource Manager

YARN Architecture

Application Master

- Once initiated, an Application Master instance creates other containers (tasks) the application needs to progress
- Each application's Application Master periodically sends heartbeat messages to the Resource Manager
- The Application Master oversees the execution of an application over its full lifespan…
  - From requesting additional containers (resources) from the Resource Manger
  - To submitting container release requests to the Node Manager on application completion

YARN Application Initiation

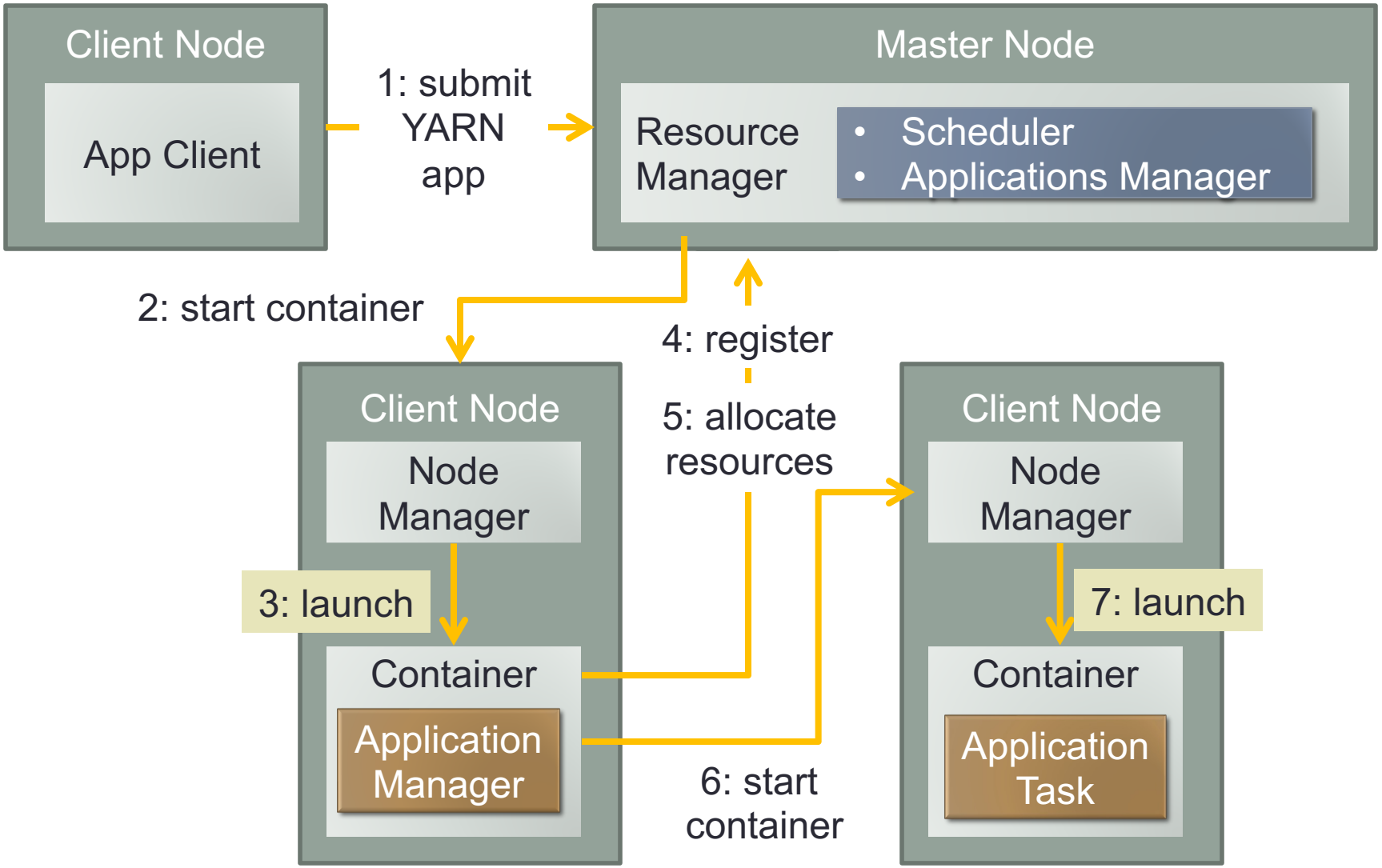
Overview

- Application execution consists of the following steps:

1. Application submission.
2. Bootstrapping the ApplicationMaster instance for the application.
3. Application execution managed by the ApplicationMaster instance

YARN Application Initiation

Details

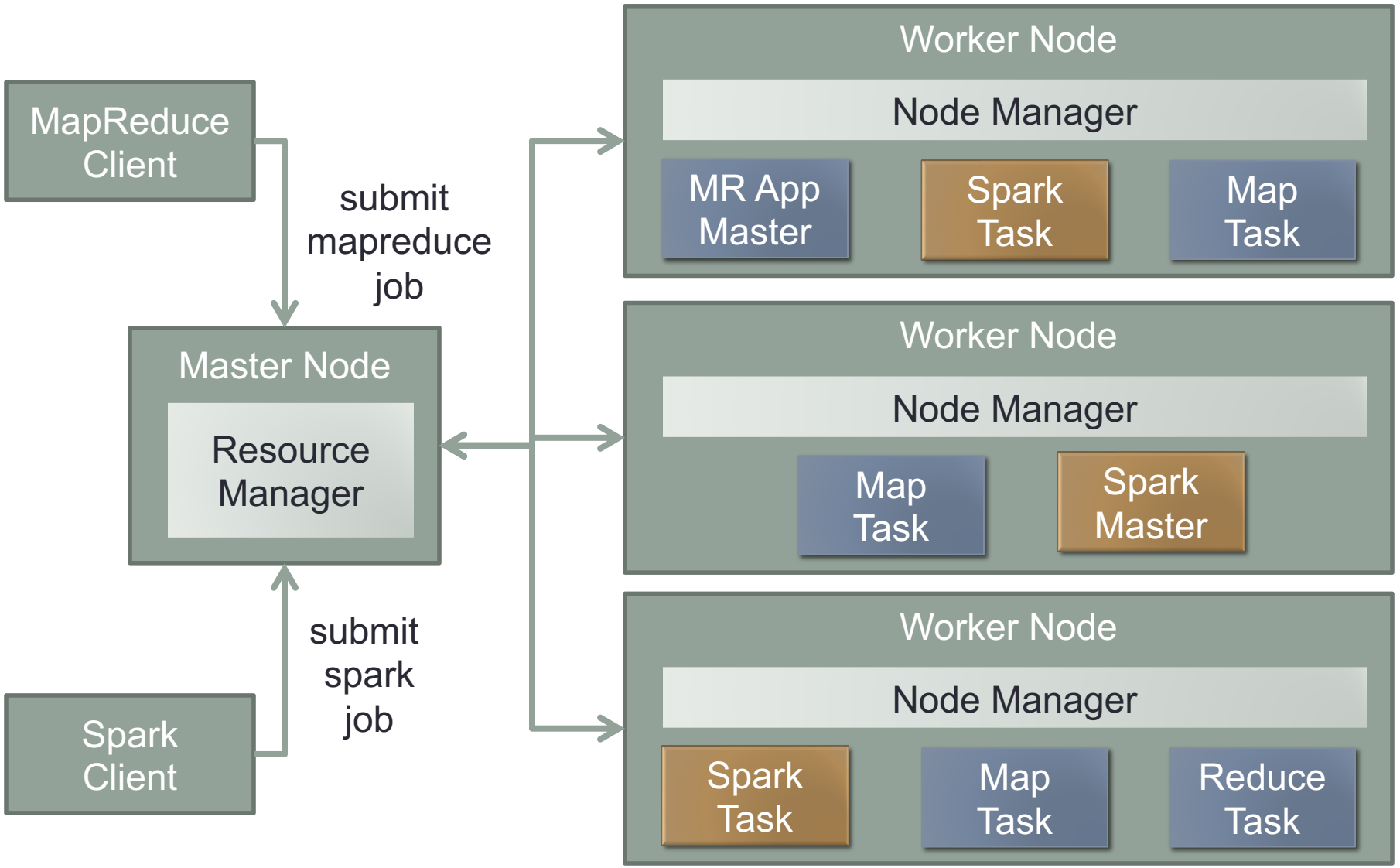


YARN Application Initiation

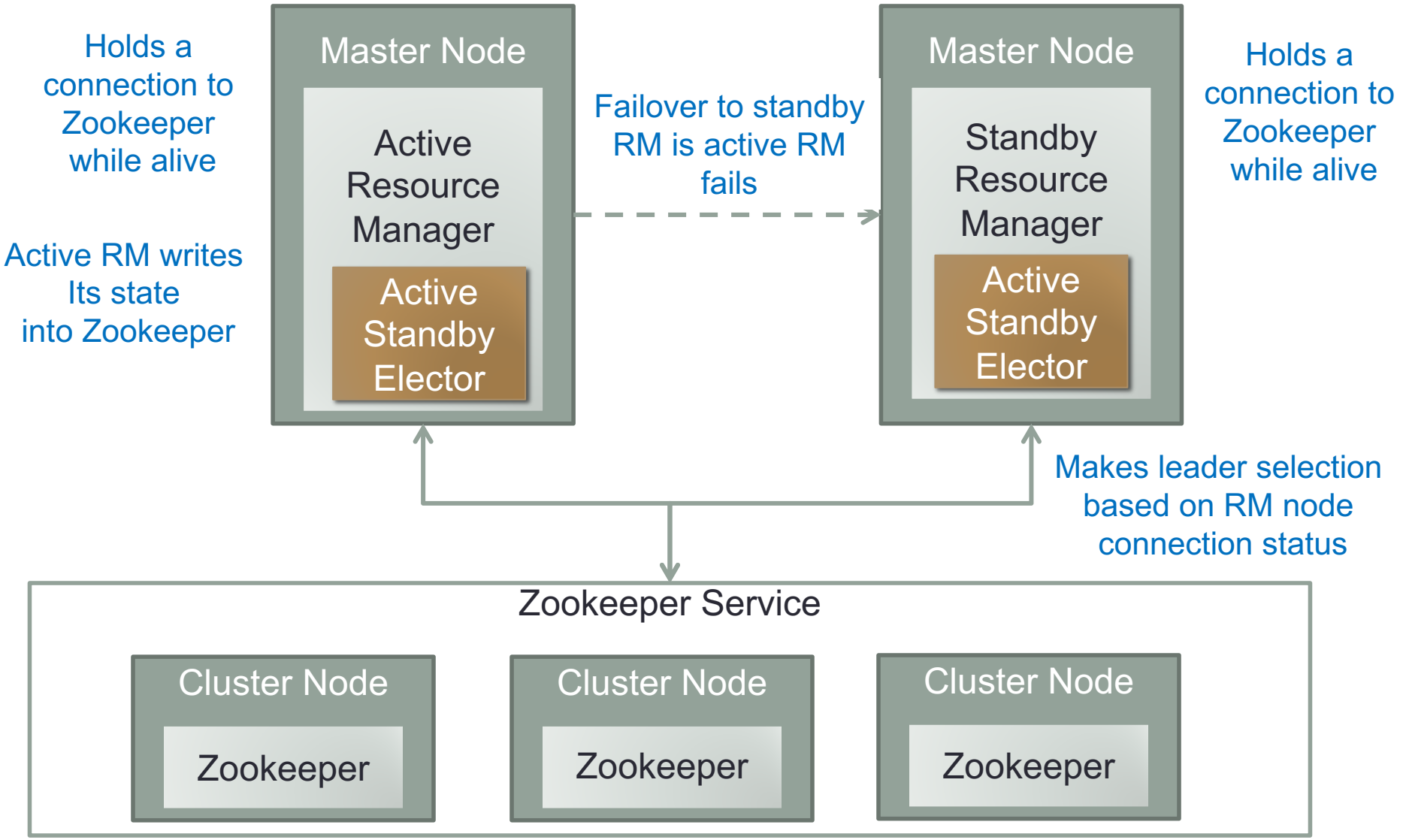
Flow

1. A client program submits the application (job) to execute
2. The Resource Manager communicates with a Node Manager
3. The Node Manager allocates a container to start an application specific Application Master
4. The Application Master, on starting up registers with the Resource Manager
5. The Application Master negotiates with the Resource Manager for one or more additional resource containers
6. On successful allocation, the Application Master contacts Node Managers to launch the containers (and tasks)
7. The Node Managers allocate containers to start application specific tasks

Multi-tenant Hadoop Cluster



Resource Manager Fault Tolerance



Resource Manager Fault Tolerance

- Realized through an Active/Standby architecture
- At any point of time, one of the RMs is Active, and one or more RMs are in Standby mode waiting to take over should anything happen to the Active
- The trigger to transition-to-active comes from either an admin (manual failover)
- Or the integrated failover controller when automatic failover is enabled
- To support automatic failover RMs embed the Zookeeper ActiveStandbyElector
  - This decides which RM should be the Active
- When Active goes down or becomes unresponsive another RM is automatically elected to be the Active which then takes over