cloudera®
Ask Bigger Questions

Can't We All Just Get Along?

Spark and Resource Management on Hadoop

Introductions

- Software engineer at Cloudera
  - MapReduce, YARN, Resource management
- Hadoop committer

cloudera®
Ask Bigger Questions

Introduction

- Spark as a first class data processing framework alongside MR and Impala
- Resource management
- What we have already
- What we need for the future

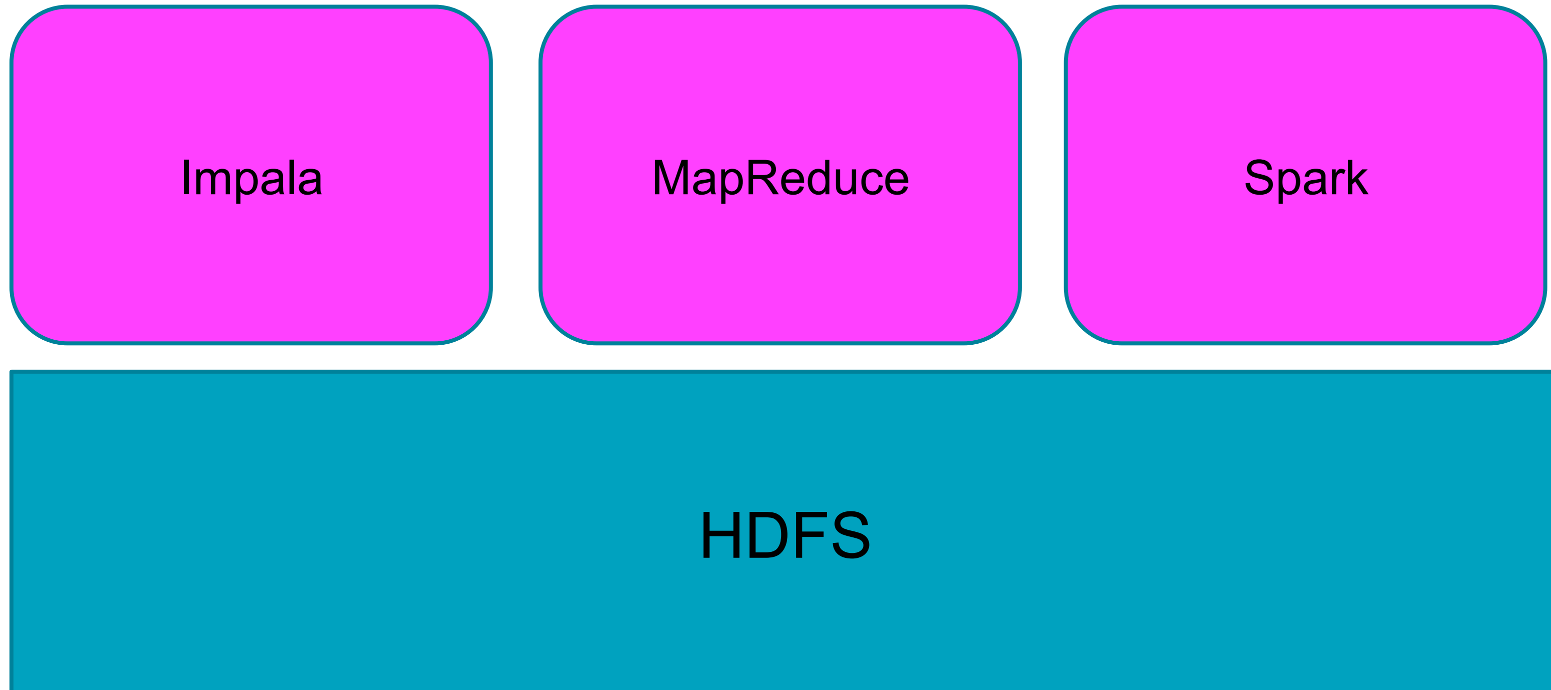
Bringing Computation to the Data

- Users want to
  - ETL a dataset with Pig and MapReduce
  - Fit a model to it with Spark
  - Have BI tools query it with Impala
- Same set of machines that hold data must also host these frameworks

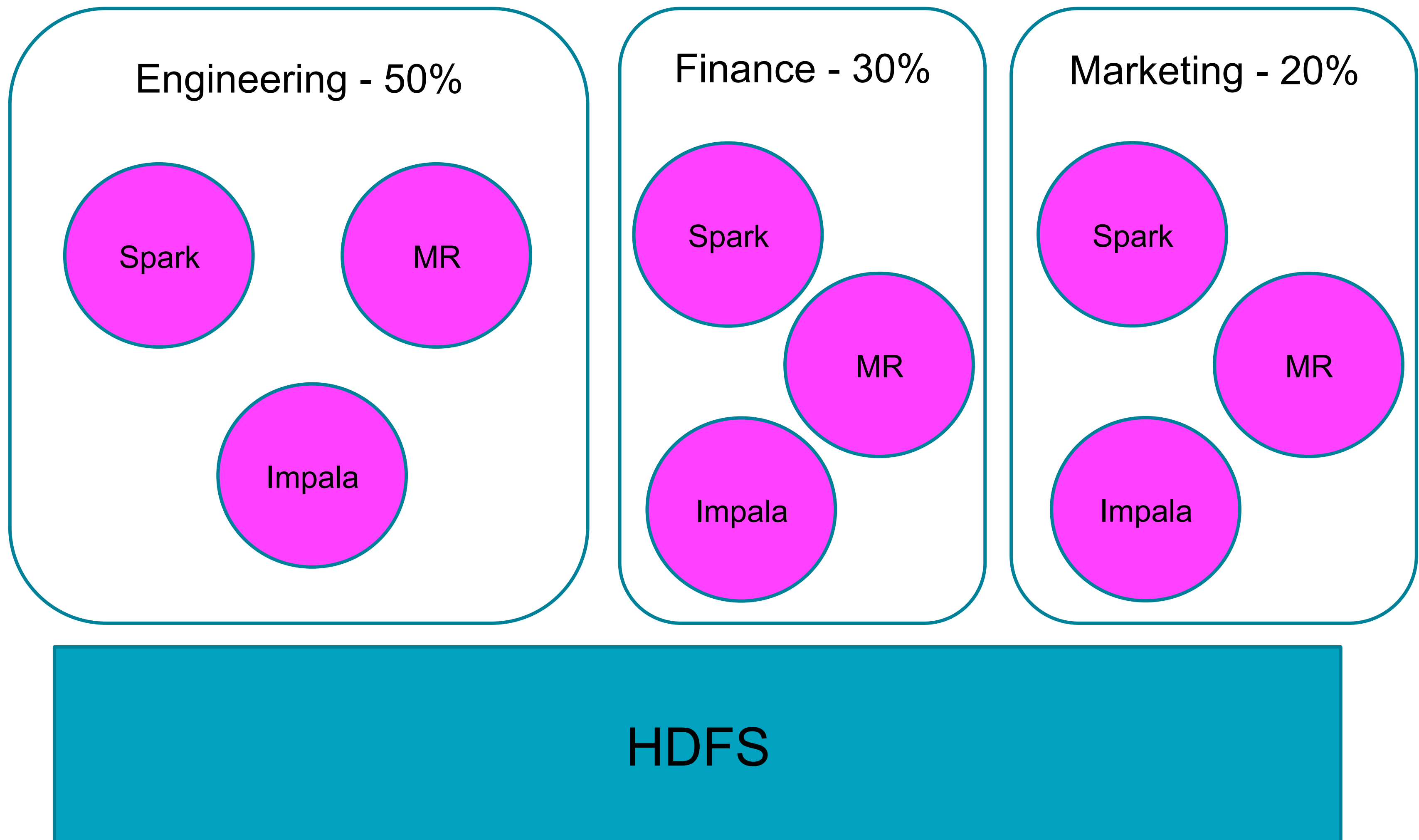
Cluster Resource Management

- Hadoop brings generalized computation to big data
- More processing frameworks
  - MapReduce, Impala, **Spark**
- Some workloads are more important than others
- A cluster has finite resources
  - Limited CPU, memory, disk and network bandwidth
- How do we make sure each workload gets the resources it deserves?

How We See It



How They Want to See It



Central Resource Management

Impala	MapReduce	Spark
YARN		
HDFS		

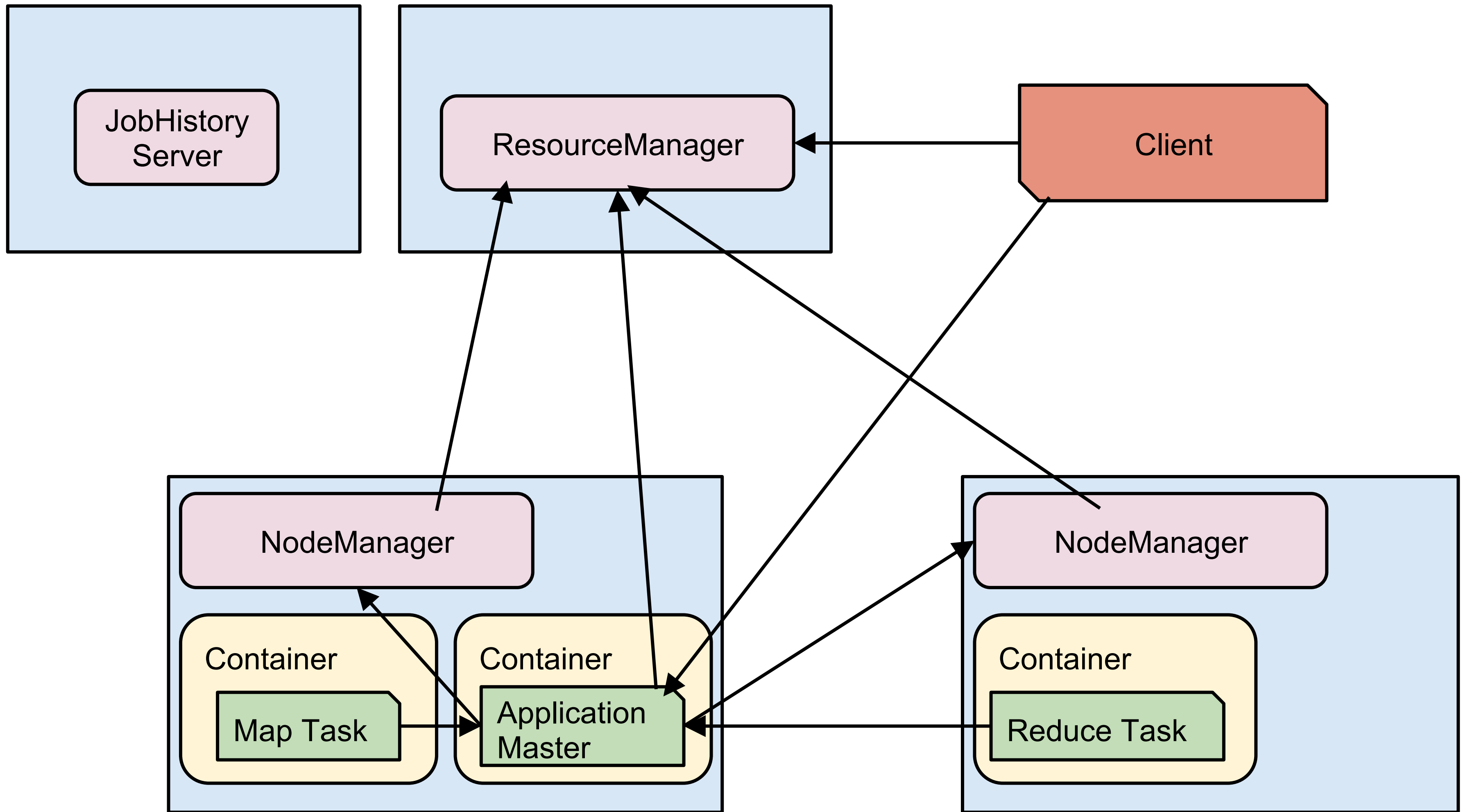
YARN

- Resource manager and scheduler for Hadoop
- “Container” is a process scheduled on the cluster with a resource allocation (amount MB, # cores)
- Each container belongs to an “Application”

YARN Application Masters

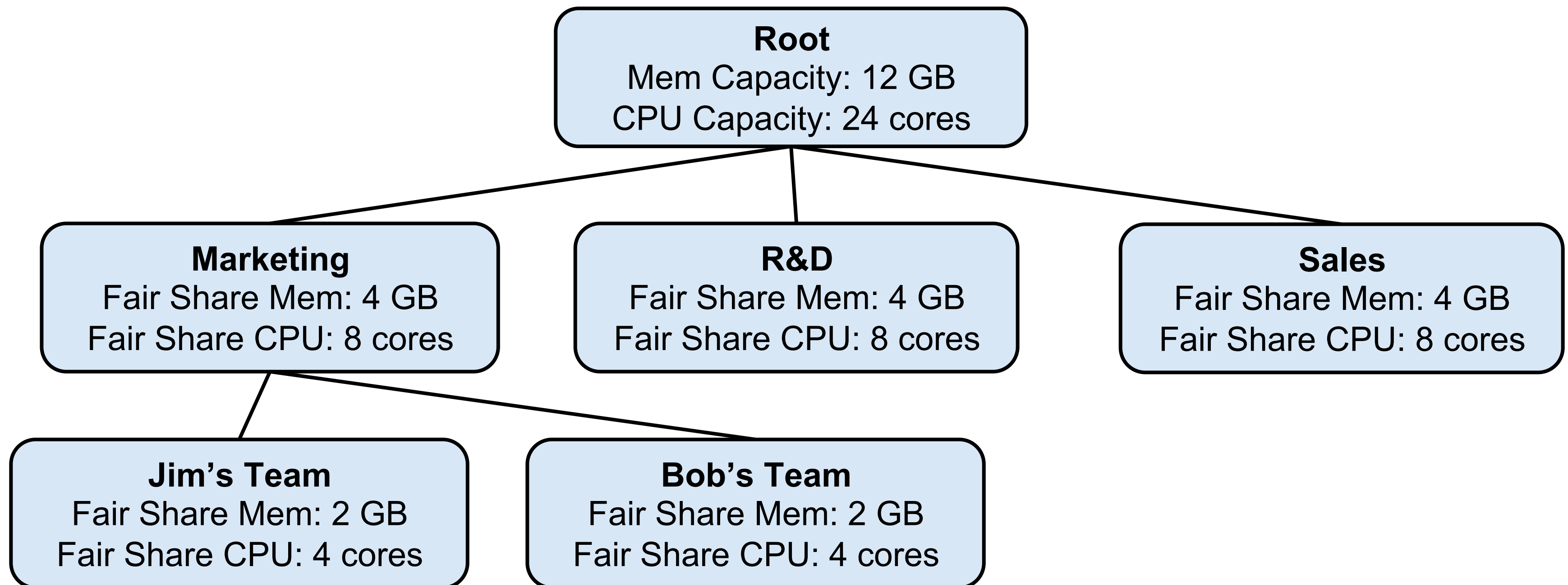
- Each YARN app has an "Application Master" (AM) process running on the cluster
- AM responsible for requesting containers from YARN
- AM creation latency is much higher than resource acquisition

YARN



YARN Queues

- Cluster resources allocated to “queues”
- Each application belongs to a queue
- Queues may contain subqueues



YARN app models

- Application master (AM) per job
  - Most simple for batch
  - Used by MapReduce
- Application master per session
  - Runs multiple jobs on behalf of the same user
  - Recently added in Tez
- AM as permanent service
  - Always on, waits around for jobs to come in
  - Used for Impala

Spark Usage Modes

Mode	Long Lived/Multiple Jobs	Multiple Users
Batch	No	No
Interactive	Yes	No
Server	Yes	Yes

Spark on YARN

- Developed at Yahoo
- Application Master per SparkContext
- Container per Spark executor
- Currently useful for Spark Batch jobs
  - Requests all resources up front

Enhancing Spark on YARN

- Long-lived sessions
  - Multiple Jobs
- Multiple Users

Long-Lived Goals

- Hang on to few resources when we’re not running work
- Use lots of the cluster (over fair share) when it’s not being used by others
- Give back resources gracefully when preempted
- Get resources quickly when we need them

Mesos Fine-Grained Mode

- Allocate static chunks of memory at Spark app start time
- Schedule CPU dynamically when running tasks

Long-Lived Approach

- A YARN application master per Spark application (SparkContext)
- Which is to say an application master per session
- One executor per application per node
- One YARN container per executor
- Executors can acquire and give back resources

Long-Lived: YARN work

- YARN-896 - long lived YARN
  - YARN not built with apps that would stick around indefinitely
  - Miscellaneous work like renewable container tokens
- YARN-1197 - resizable containers

Long-Lived: Spark Work

- YARN fine-grained mode
- Changes to support adjusting resources in Spark AM
- Memory?

The Memory Problem

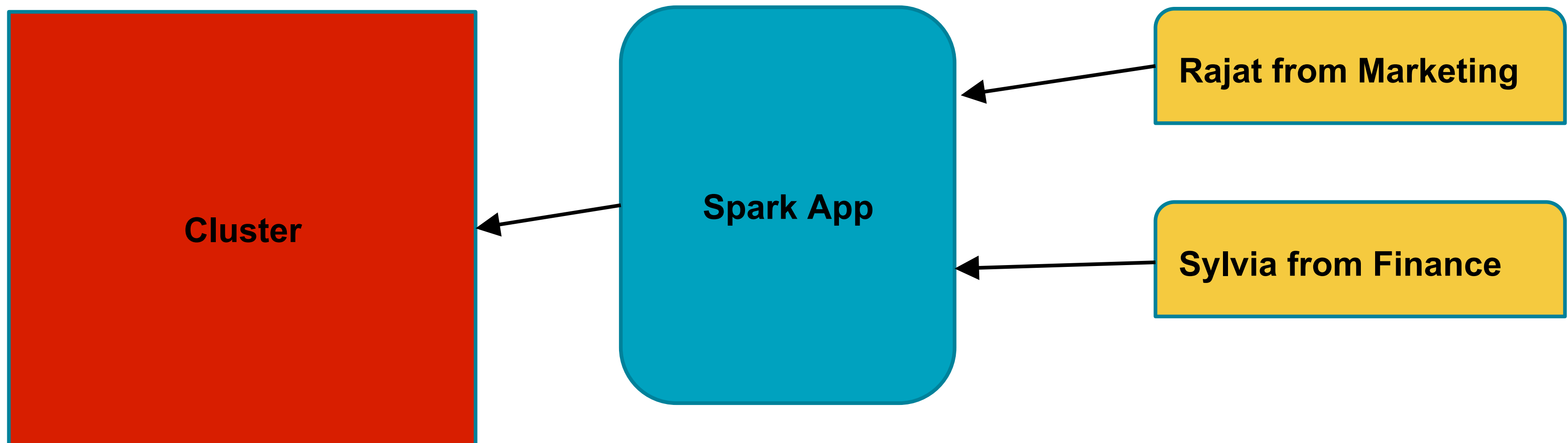
- We want to be able to have memory allocations preempted and keep running
- RDDs stored in JVM memory
- JVMs don’t give back memory

The Memory Solutions

- Rewrite Spark in C++
- Off-heap cache
  - Hold RDDs in executor processes in off-heap byte buffers
  - These can be freed and returned to the OS
- Tachyon
  - Executor processes don't hold RDDs
  - Store data in Tachyon
  - Punts off-heap problem to Tachyon
  - Has other advantages, like not losing data when executor crashes

Multiple User Challenges

- A single Spark application wants to run work on behalf of multiple parties
- Applications are typically billed to a single queue
- We'd want to bill jobs to different queues

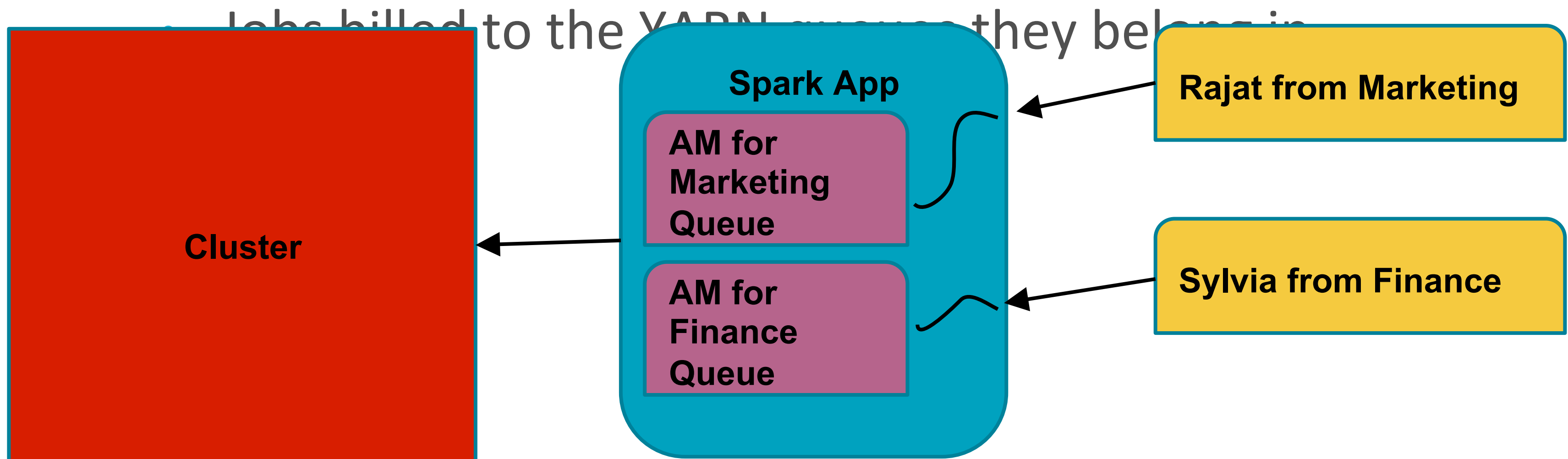


Multiple Users with Spark Fair Scheduler

- Full-features Fair Scheduler within a Spark Application
- Two level scheduling
- Difficult to share dynamically between Spark and other frameworks

Multiple Users with Impala

- Impala has same exact problem
- Solution: Llama (Low Latency Application MAster)
  - Adapter between YARN and Impala
  - Runs multiple AMs in a single process
  - Submits resource requests on behalf of relevant AM
  - Jobs billed to the YARN queues they belong in



Spark
Other Hadoop processing frameworks