Spark on YARN

Agenda

- YARN - Introduction
- Need for YARN
- OS Analogy
- Why run Spark on YARN
- YARN Architecture
- Modes of Spark on YARN
- Internals of Spark on YARN
- Recent developments
- Road ahead
- Hands-on

YARN

- Yet another resource negotiator.
- a general-purpose, distributed, application management framework.

Need for YARN

Hadoop 1.0

- Single use system
- Capable of running only MR

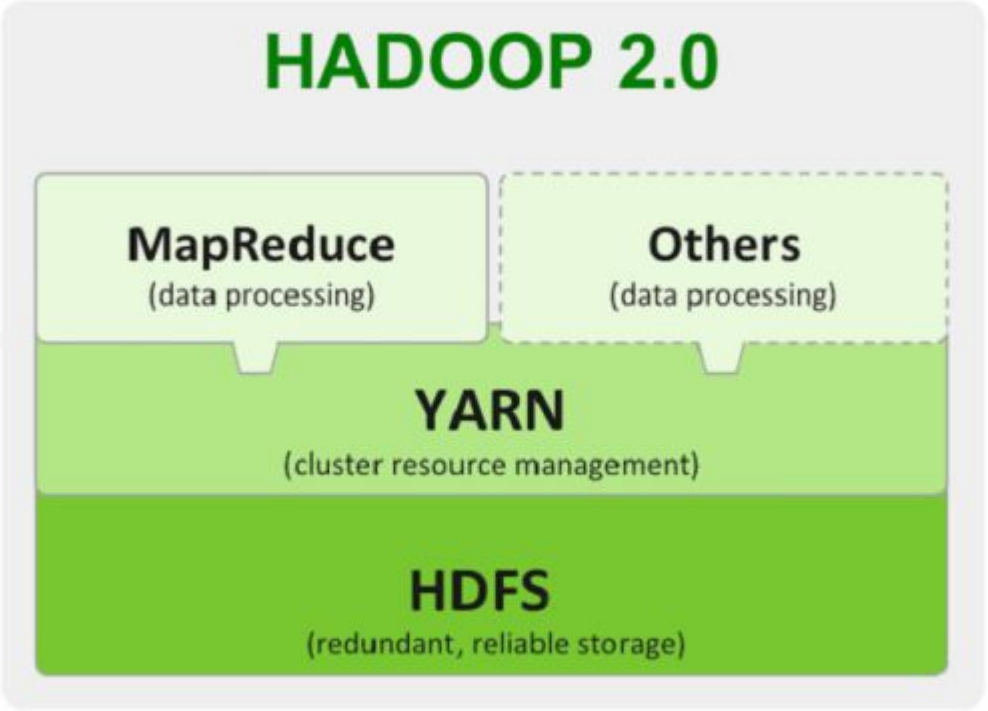
Need for YARN

- Scalability
  - 2009 – 8 cores, 16GB of RAM, 4x1TB disk
  - 2012 – 16+ cores, 48-96GB of RAM, 12x2TB or 12x3TB of disk.
- Cluster utilization
  - distinct map slots and reduce slots
- Supporting workloads other than MapReduce
  - MapReduce is great for many applications, but not everything.

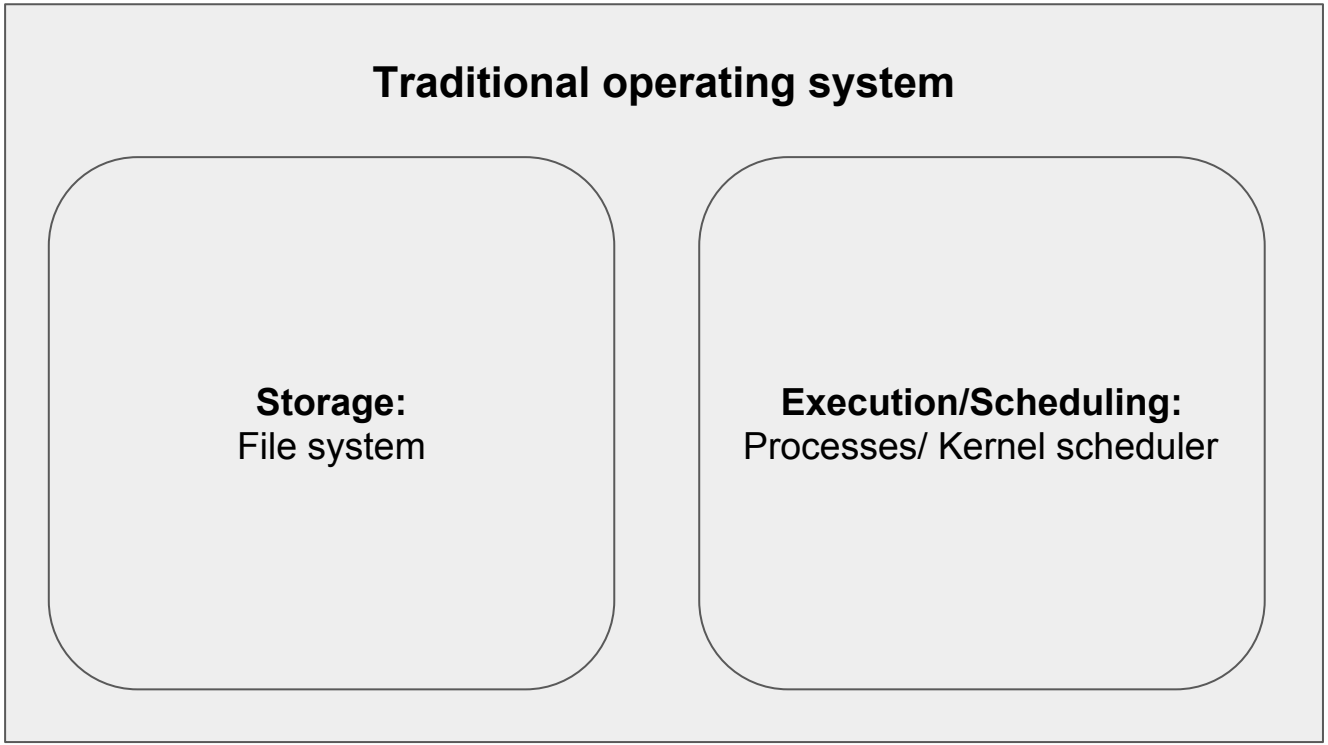
Need for YARN

Hadoop 2.0

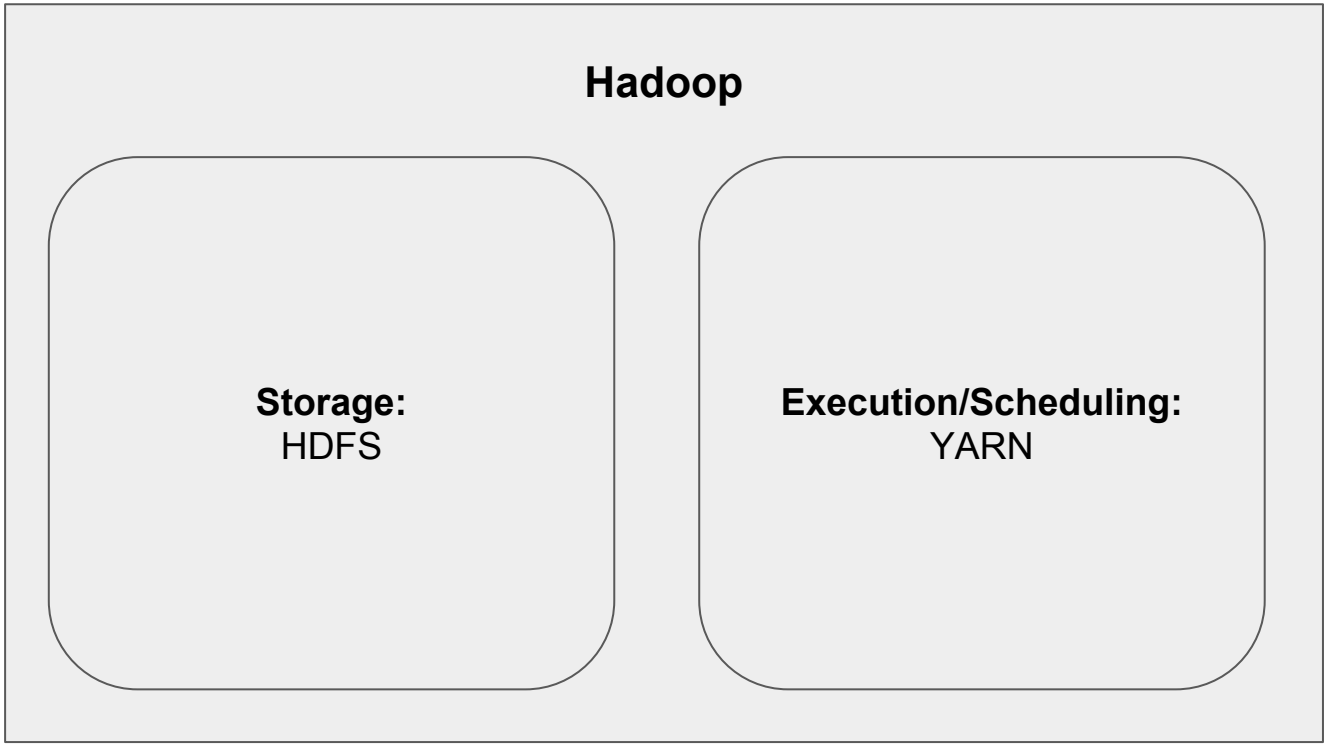
- Multi purpose platform
- Capable of running apps other than MR



OS analogy


Traditional operating system
Storage:
File system
Execution/Scheduling:
Processes/ Kernel scheduler

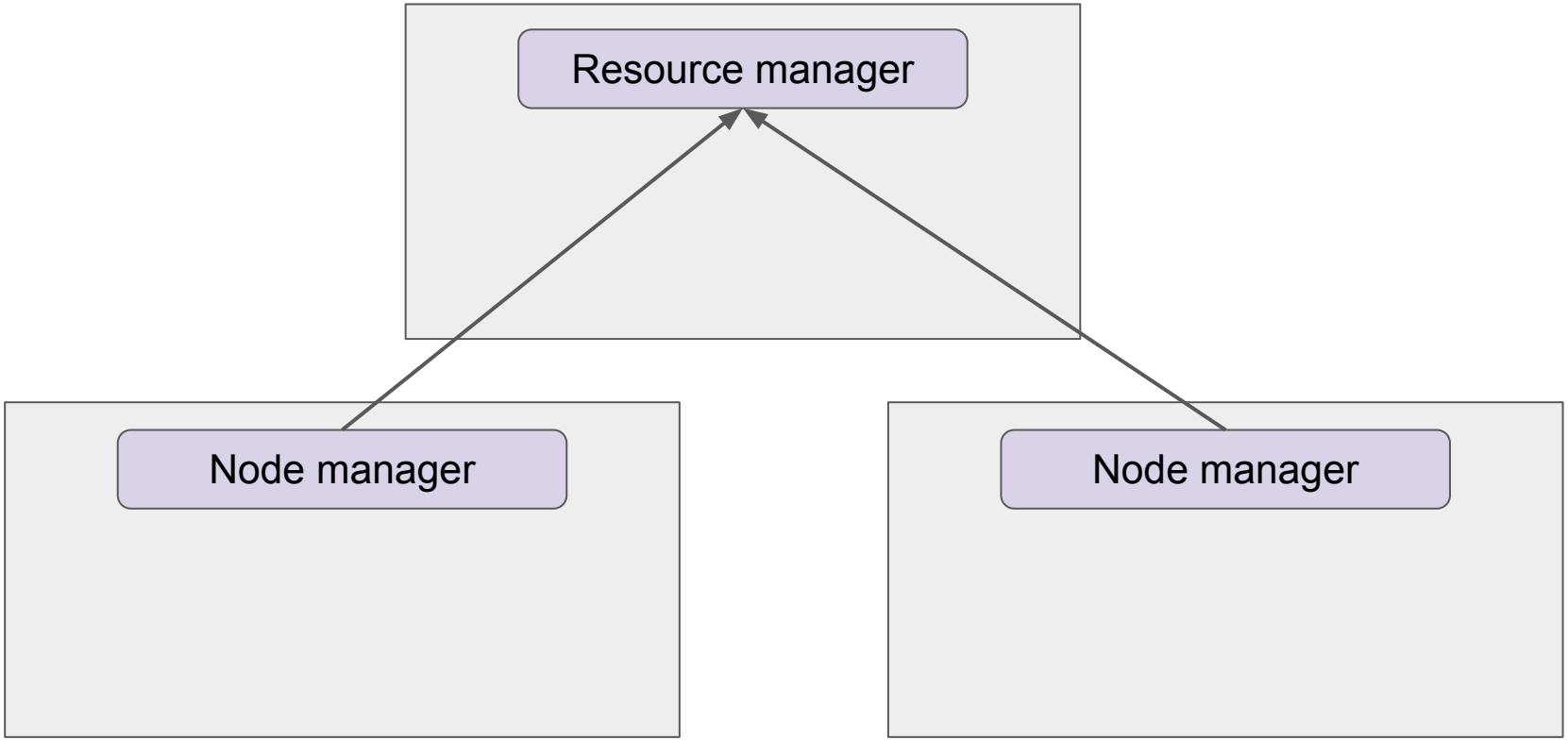
OS analogy



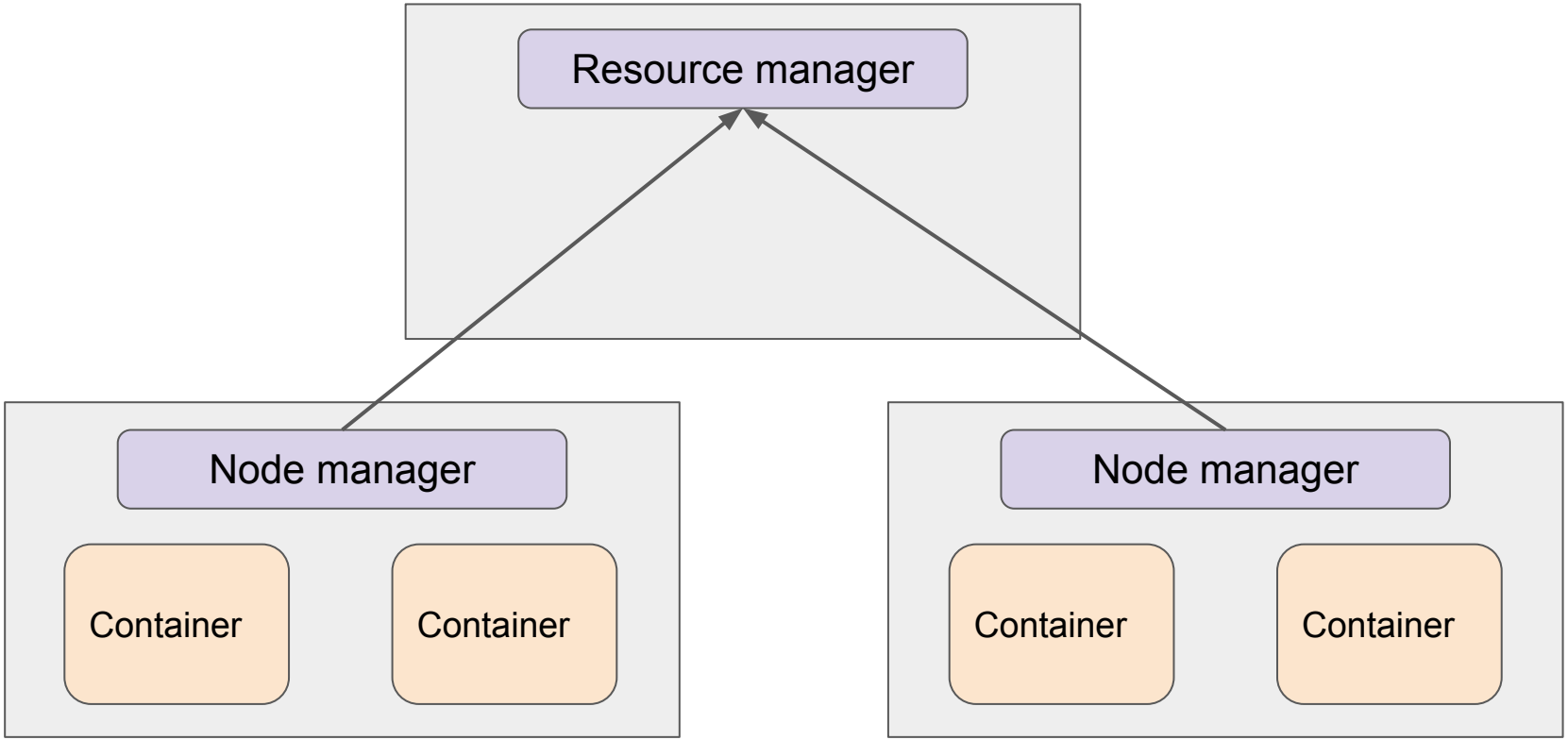
Why run Spark on YARN

- Leverage existing clusters
- Data locality
- Dynamically sharing the cluster resources between different frameworks.
- YARN schedulers can be used for categorizing, isolating, and prioritizing workloads.
- Only cluster manager for Spark that supports security

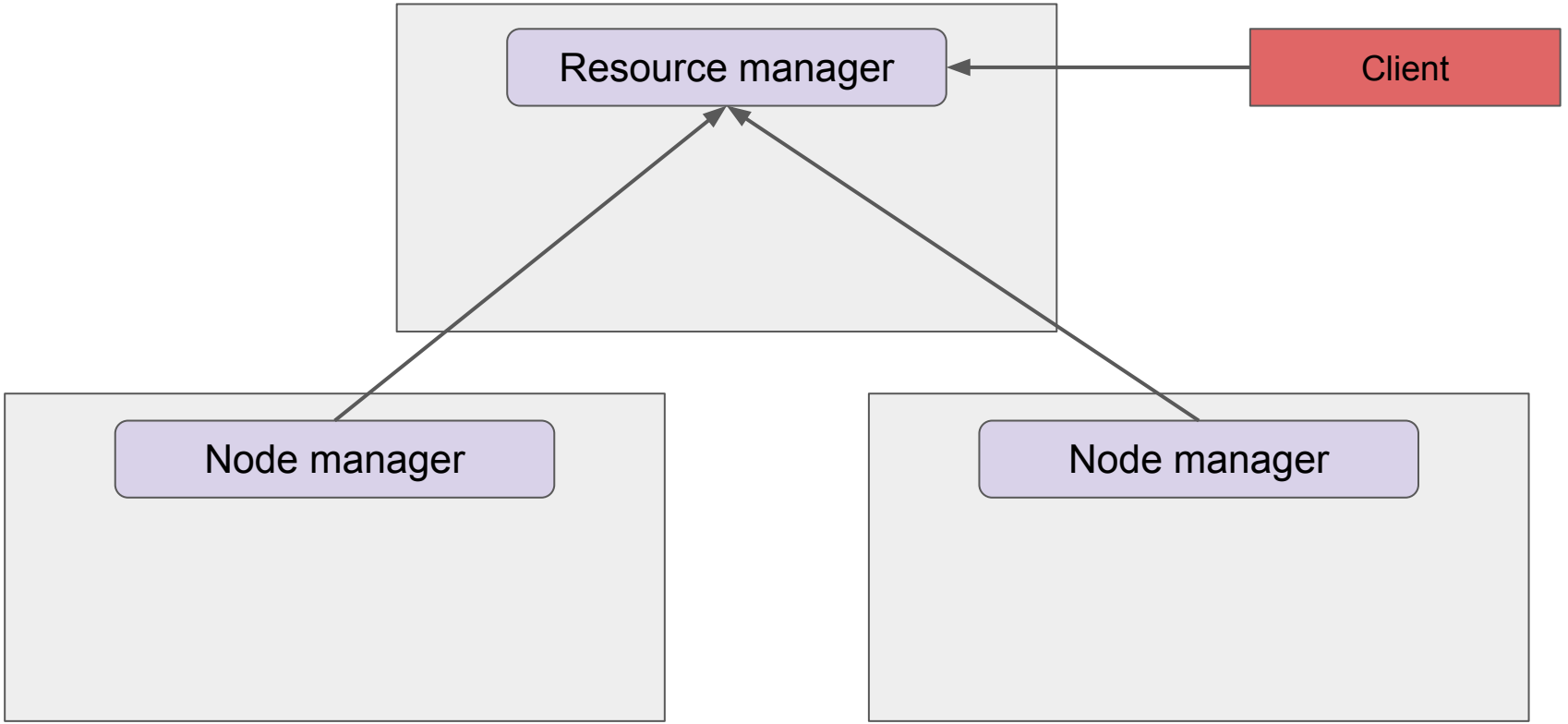
YARN architecture



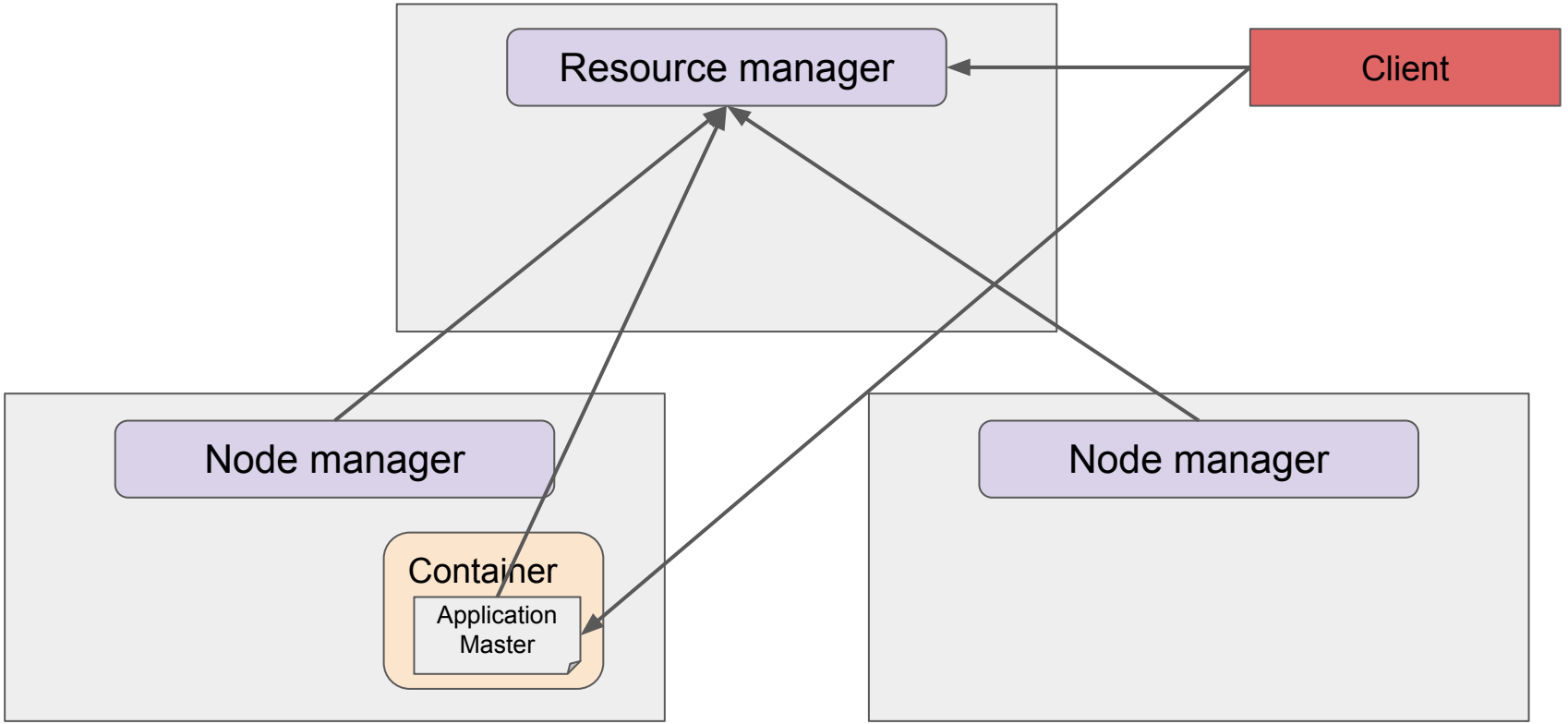
YARN architecture



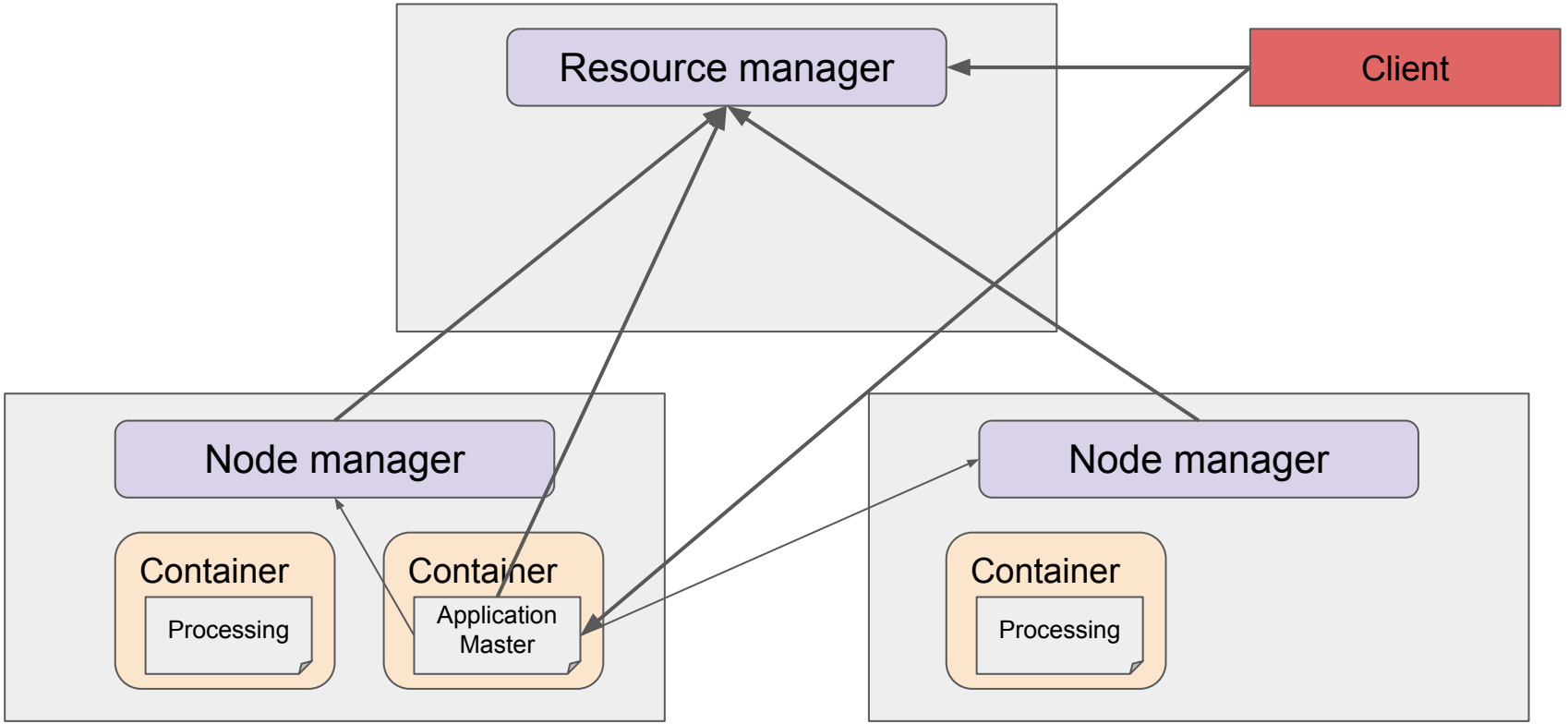
YARN architecture



YARN architecture

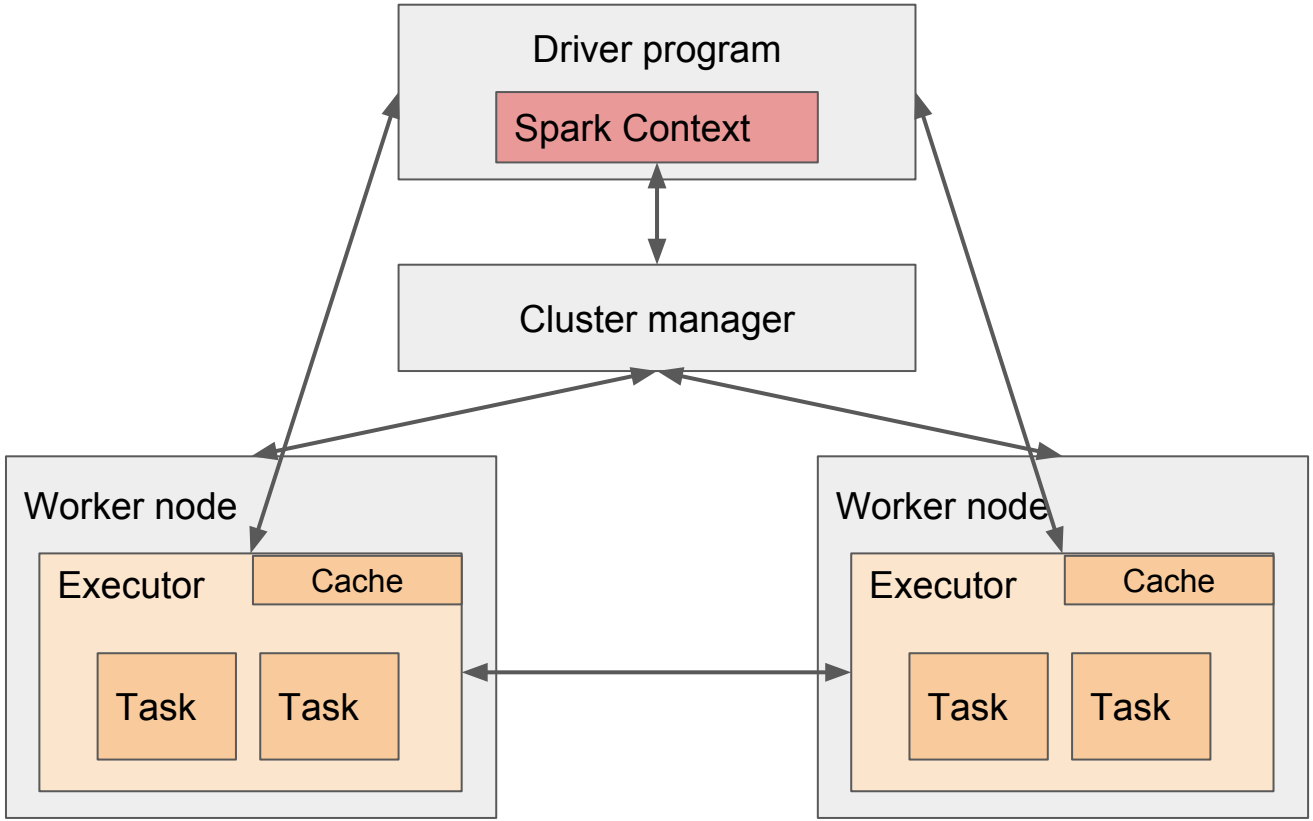


YARN architecture



Running Spark on YARN

Spark architecture



Spark architecture

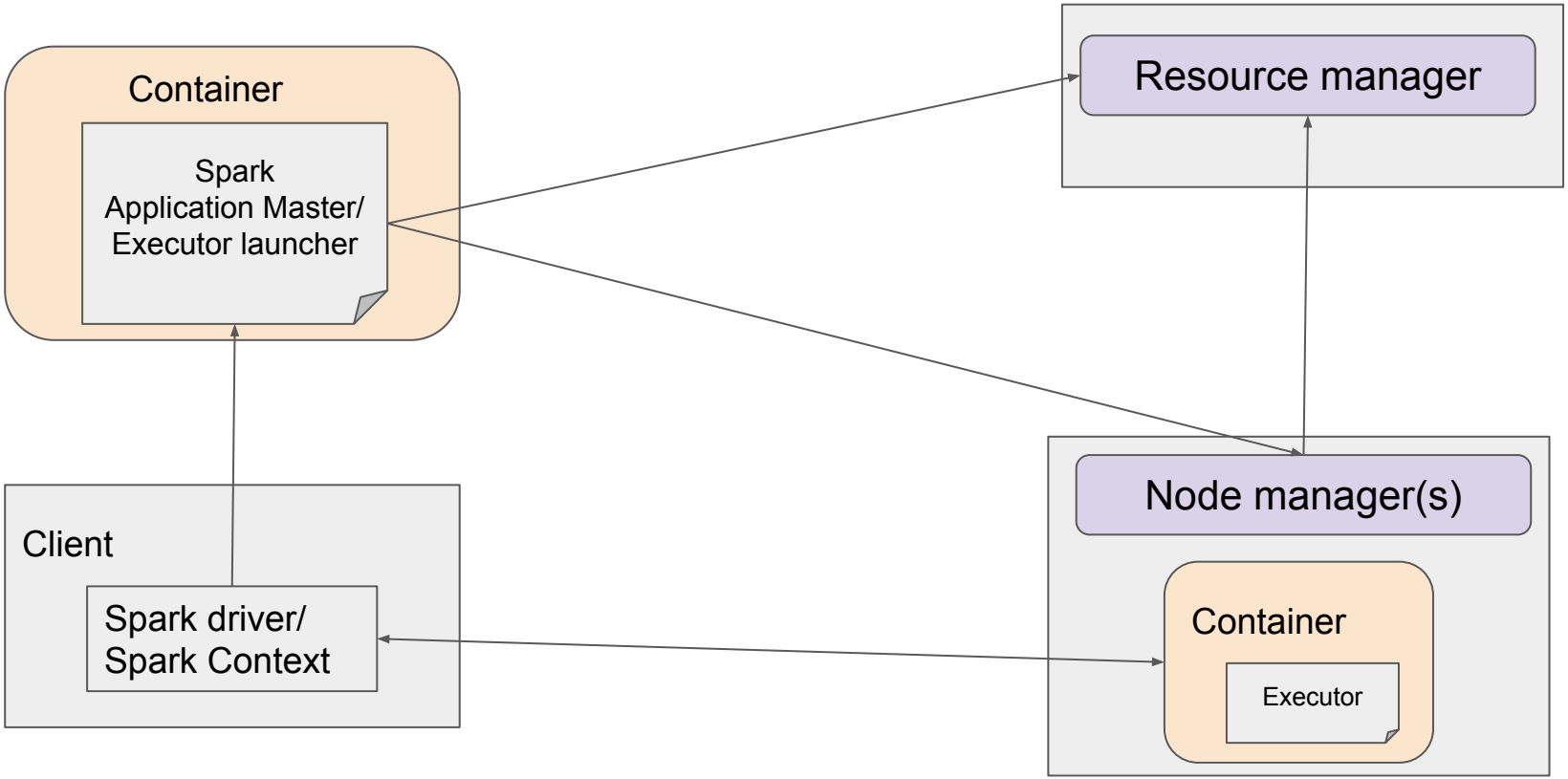
- **Driver Program** is responsible for managing the job flow and scheduling tasks that will run on the executors.
- **Executors** are processes that run computation and store data for a Spark application.
- **Cluster Manager** is responsible for starting executor processes and where and when they will be run. Spark supports pluggable cluster manager, it supports

  Example: YARN, Mesos and “standalone” cluster manager

Modes on Spark on YARN

- YARN-Client Mode
- YARN-Cluster Mode

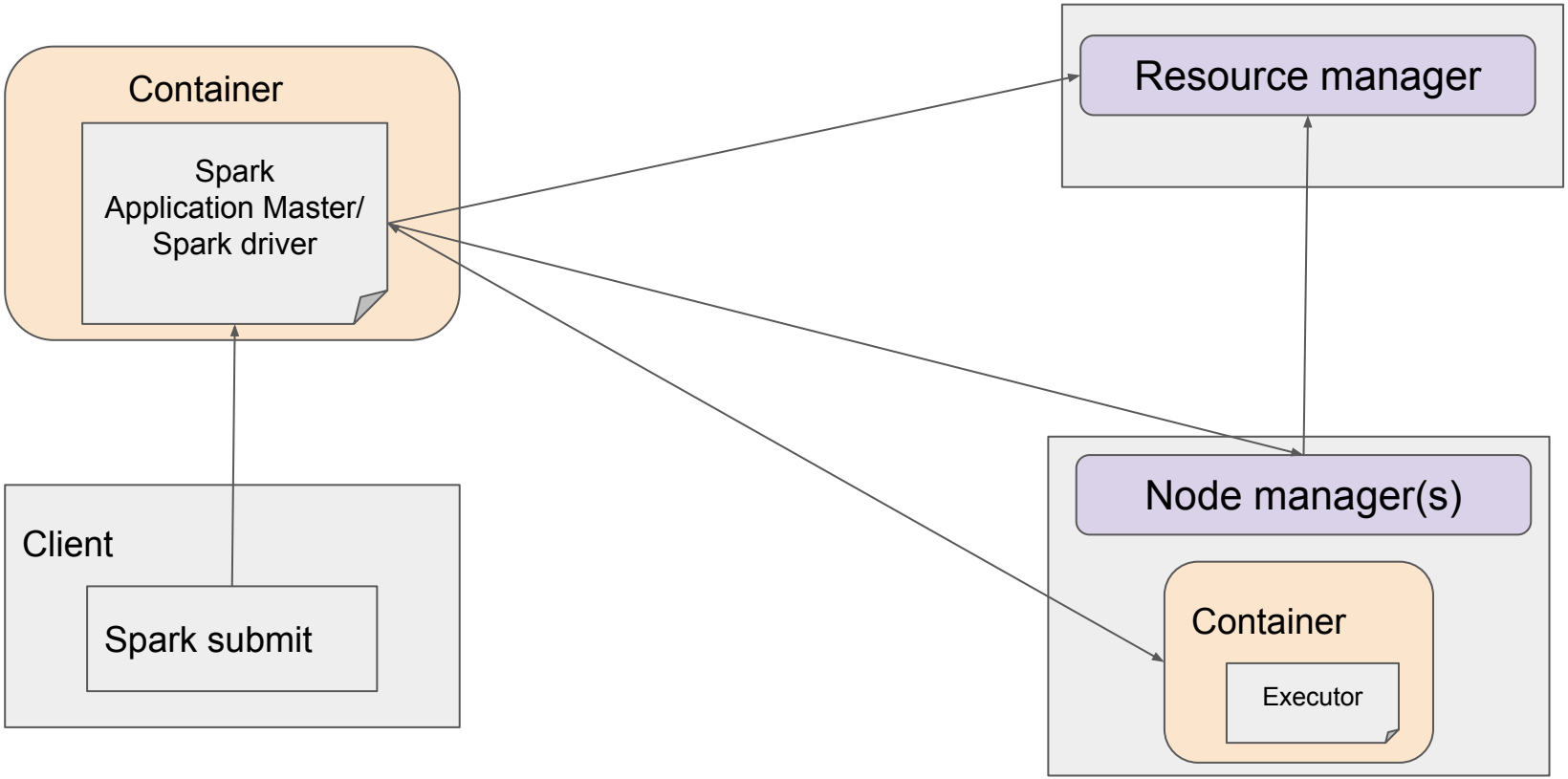
YARN client mode



YARN client mode

- Driver runs in the client process, and the application master is only used for requesting resources from YARN.
- Used for interactive and debugging uses where you want to see your application’s output immediately (on the client process side).

YARN cluster mode



YARN cluster mode

- In yarn-cluster mode, the Spark driver runs inside an application master process which is managed by YARN on the cluster, and the client can go away after initiating the application.
- Yarn-cluster mode makes sense for production jobs.

Concurrency vs Parallelism

- Concurrency is about dealing with lots of things at once.
- Parallelism is about doing lots of things at once.

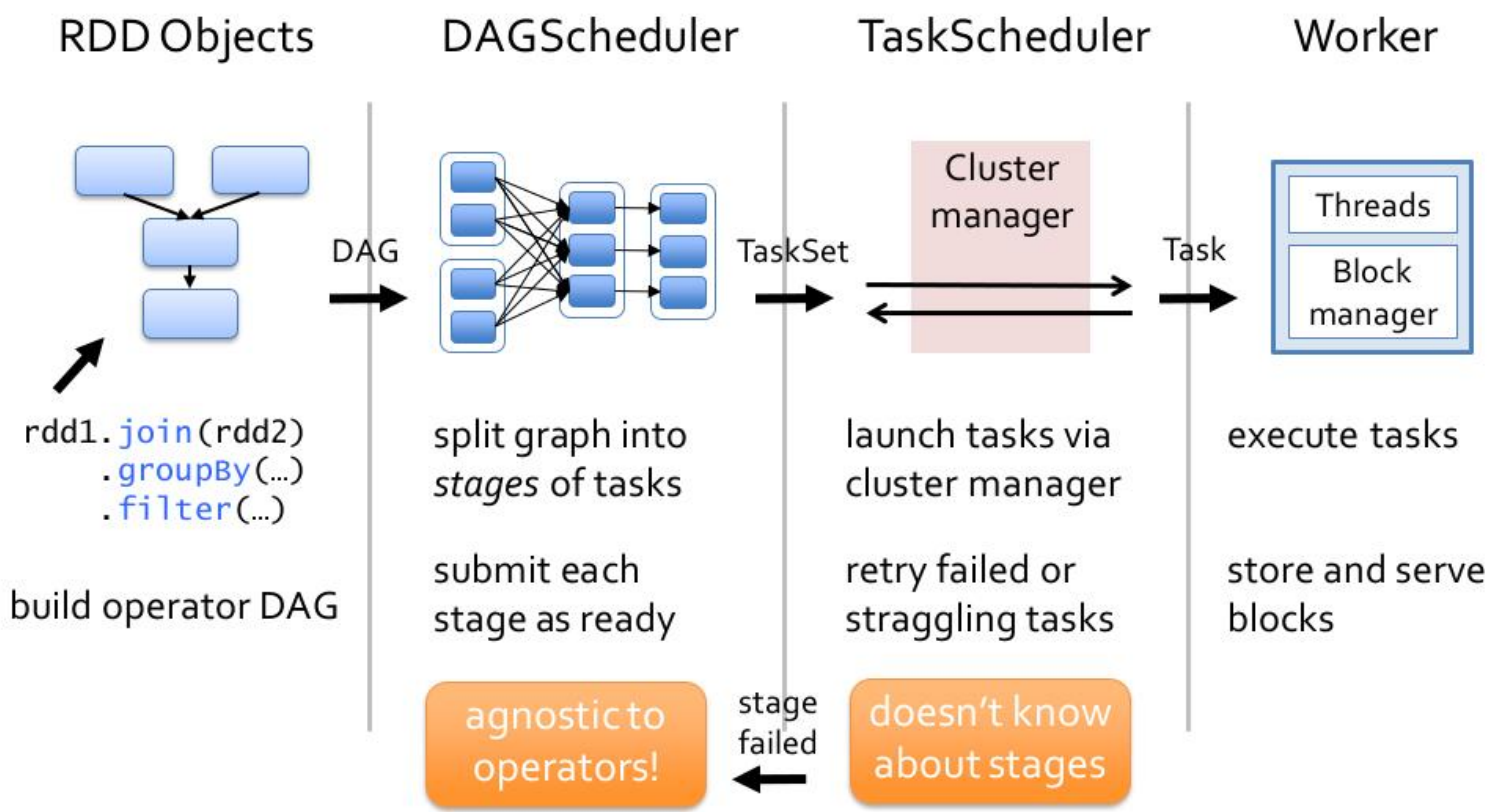
Akka

- Follows Actor model
  - Keep mutable state internally and communicate through async messages
- Actors

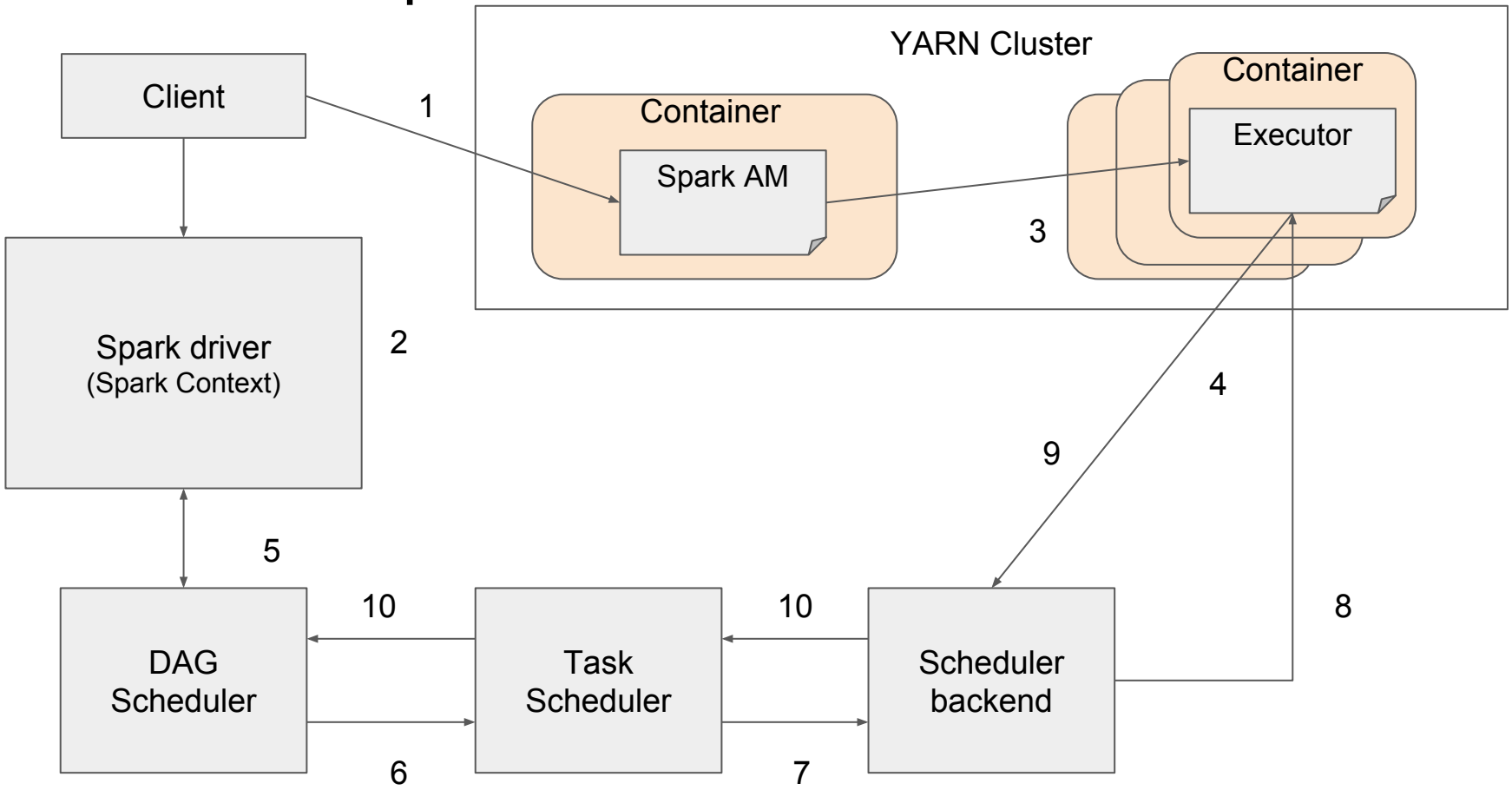
  Treat Actors like People. People who don't talk to each other in person. They just talk through mails.

  - Object
  - Runs in its own thread
  - Messages are kept in queue and processed in order

Internals of Spark



Internals of Spark on YARN



Internals of Spark on YARN

1. Requests container for the AM and launches AM in the container
2. Creates SparkContext (inside AM / inside Client).
   This internally creates a DAG Scheduler, Task scheduler and Scheduler backend.
   Creates an Akka actor system.
3. Application master based on the required resources will request for the containers. Once it get the containers it runs executor process in the container.
4. The executor process when it comes up registers with the Schedulerbackend through Akka.
5. When few lines of code has to be run on the cluster. RDD runJob method calls the DAG scheduler to create a DAG of tasks.

Internals of Spark on YARN

6. Set of tasks which is capable of running in parallel is sent to the Task Scheduler in the form of TaskSet.
7. Task scheduler in turn will contact the Schedulerbackend to run the tasks on the executor.
8. Scheduler backend which keeps track of running executors and its statuses, will schedule tasks on executors
9. Task output if any are sent through heartbeats to Schedulerbackend/
10. SchedulerBackend passes the task output onto the Task and DAG scheduler which could make use of that output.

Recent developments

- Dynamic resource allocation
  - No need to specify number of executors
  - Application grows and shrinks based on outstanding task count
  - Need to specify other things
- Data locality
  - Allocate executors close to data
  - SPARK-4352
- Cached RDDs
  - Keep executors around

Road ahead

- Making dynamic allocation better
  - Reduce allocation latency
  - Handle cached RDDs
- Simplified allocation
- Encrypt shuffle files
- File distribution
  - Replace HTTP with RPC