@deanwampler

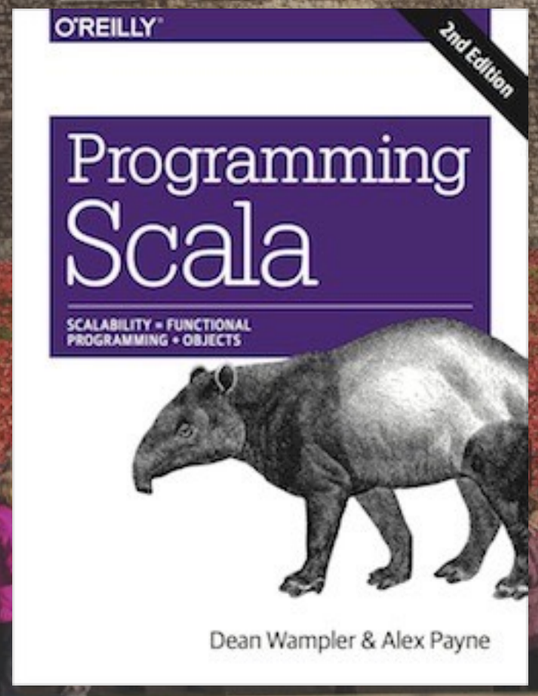
Spark on Mesos

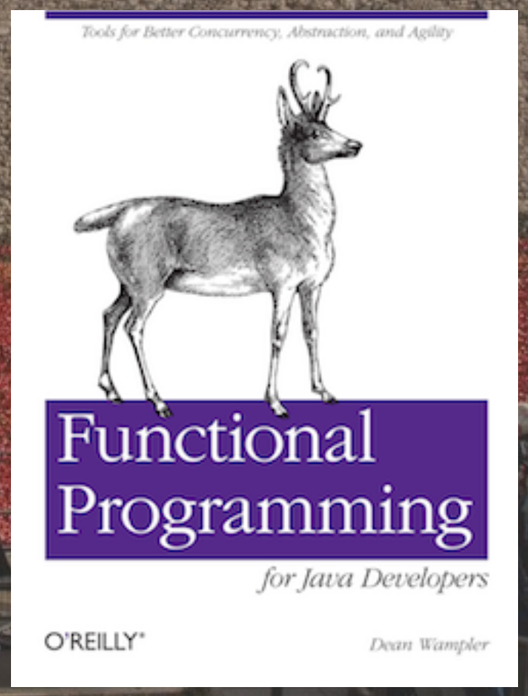
Typesafe

Tuesday, October 20, 15

Dean Wampler

dean.wampler@typesafe.com
polyglotprogramming.com/talks
@deanwampler





Tuesday, October 20, 15

About me. You can find this presentation and others on Big Data and Scala at polyglotprogramming.com.


Why Mesos?

Something Familiar

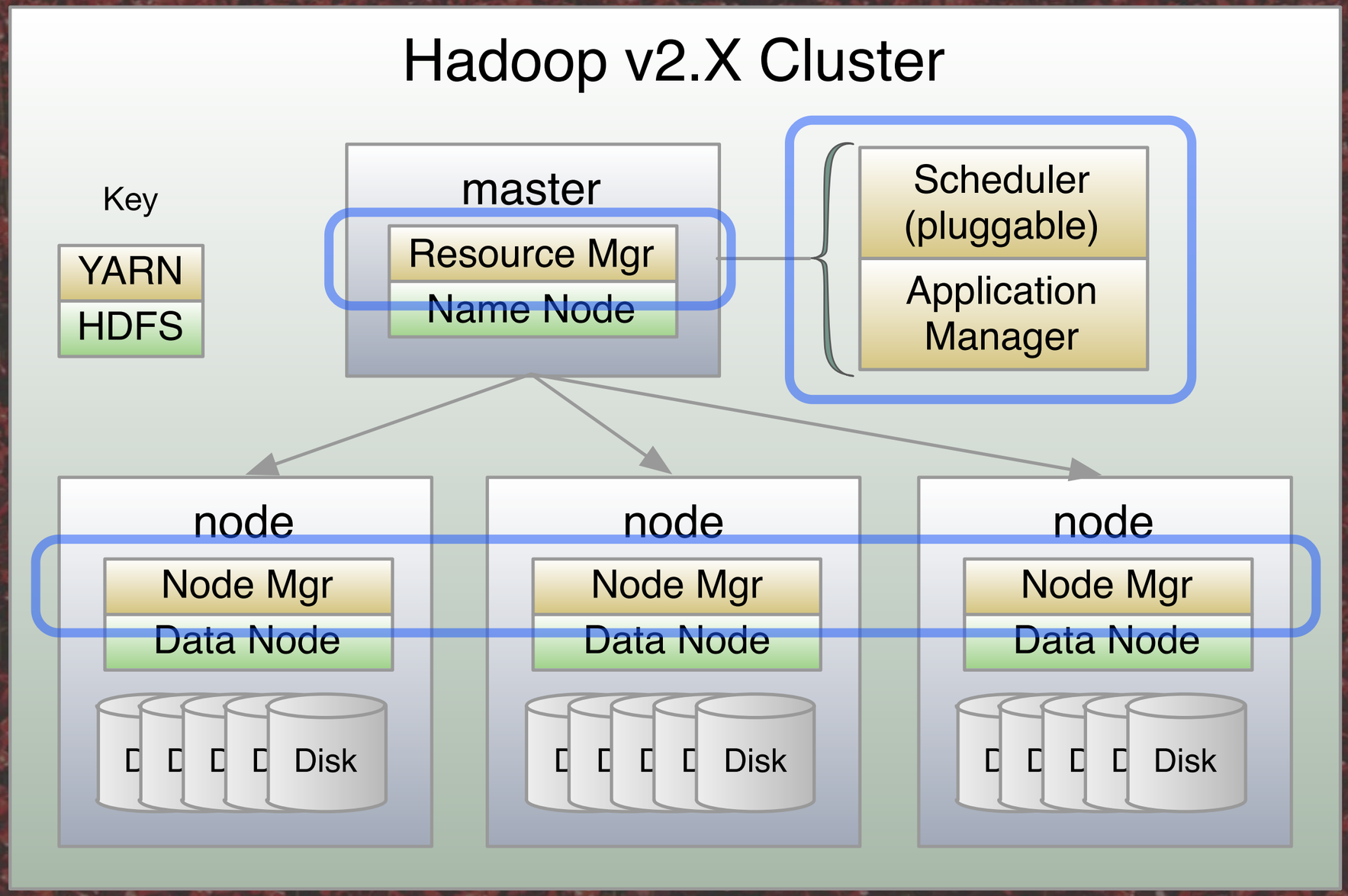
YARN

(Yet Another Resource Negotiator)

Tuesday, October 20, 15

Let’s discuss Mesos by analog with something you may already know, YARN.

Resource and Node Managers



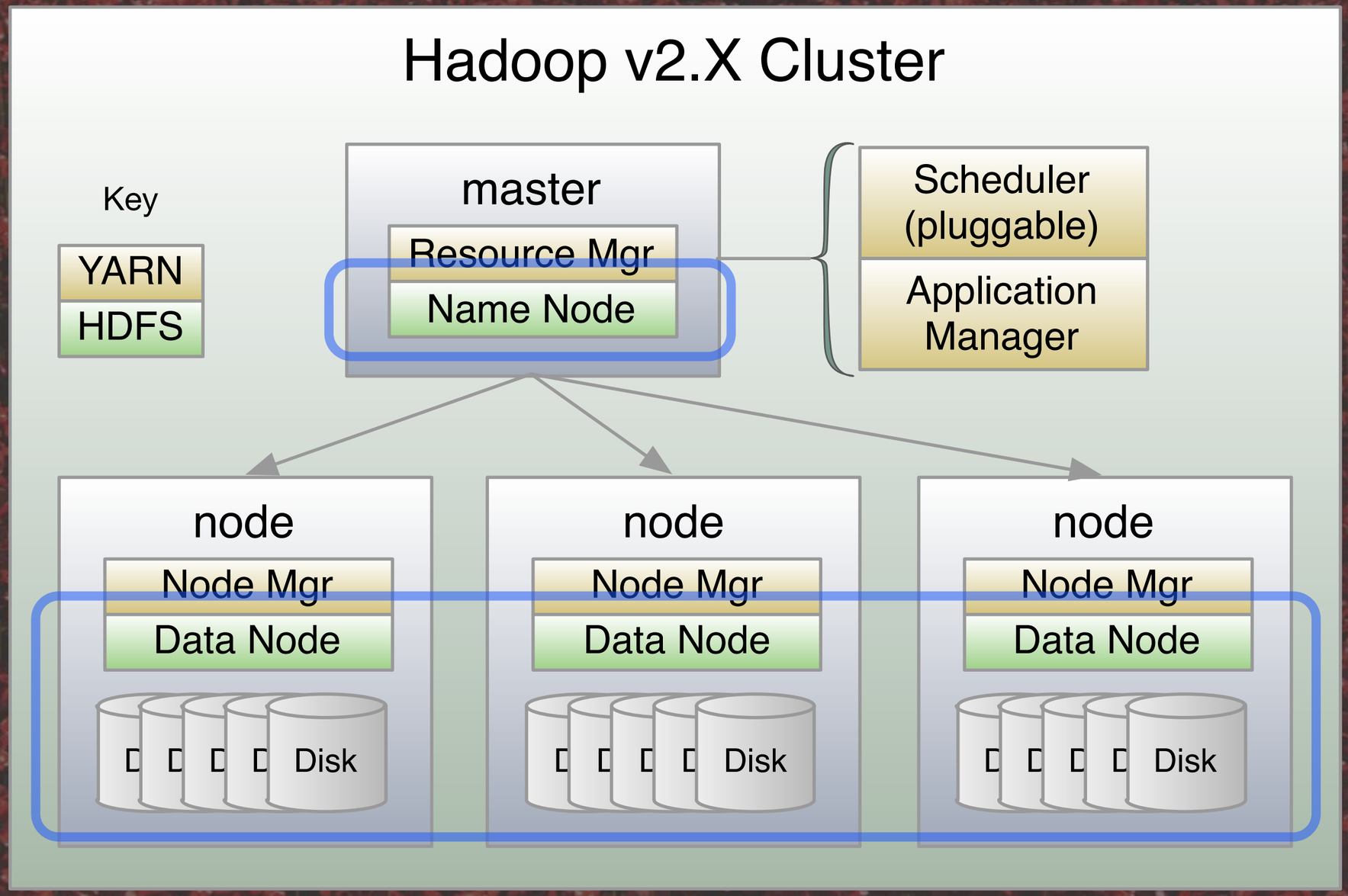
Tuesday, October 20, 15

Hadoop 2 uses YARN to manage resources via the master Resource Manager, which includes a pluggable job Scheduler and an Application Manager (where user jobs reside). It coordinates with the Node Manager on each node to schedule jobs and provide resources.

See “Apache Hadoop YARN, Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2” (Addison-Wesley) and http://hadoop.apache.org/docs/r2.3.0/hadoop-yarn/hadoop-yarn-site/YARN.html for the gory details.

Master failover to standbys can be managed by Zookeeper (not shown).

HDFS: Name Node and Data Nodes



Tuesday, October 20, 15

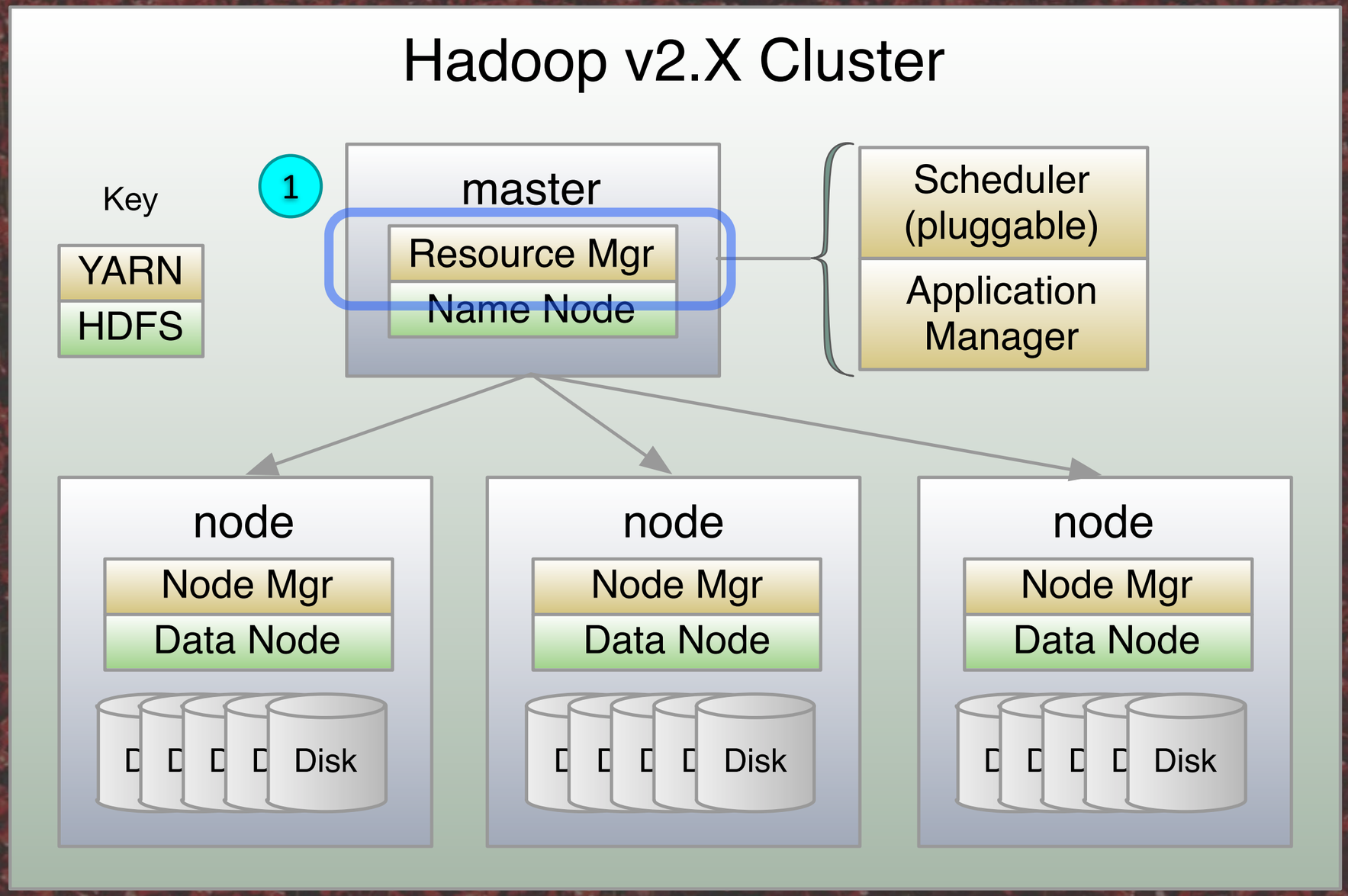
Hadoop 2 clusters federate the Name node services that manage the file system, HDFS. They provide horizontal scalability of file-system operations and resiliency when service instances fail. The data node services manage individual blocks for files. It's important to note that HDFS is NOT managed by YARN.

Job Submission

Tuesday, October 20, 15

At a high-level, let’s walk through the job submission process managed by YARN.

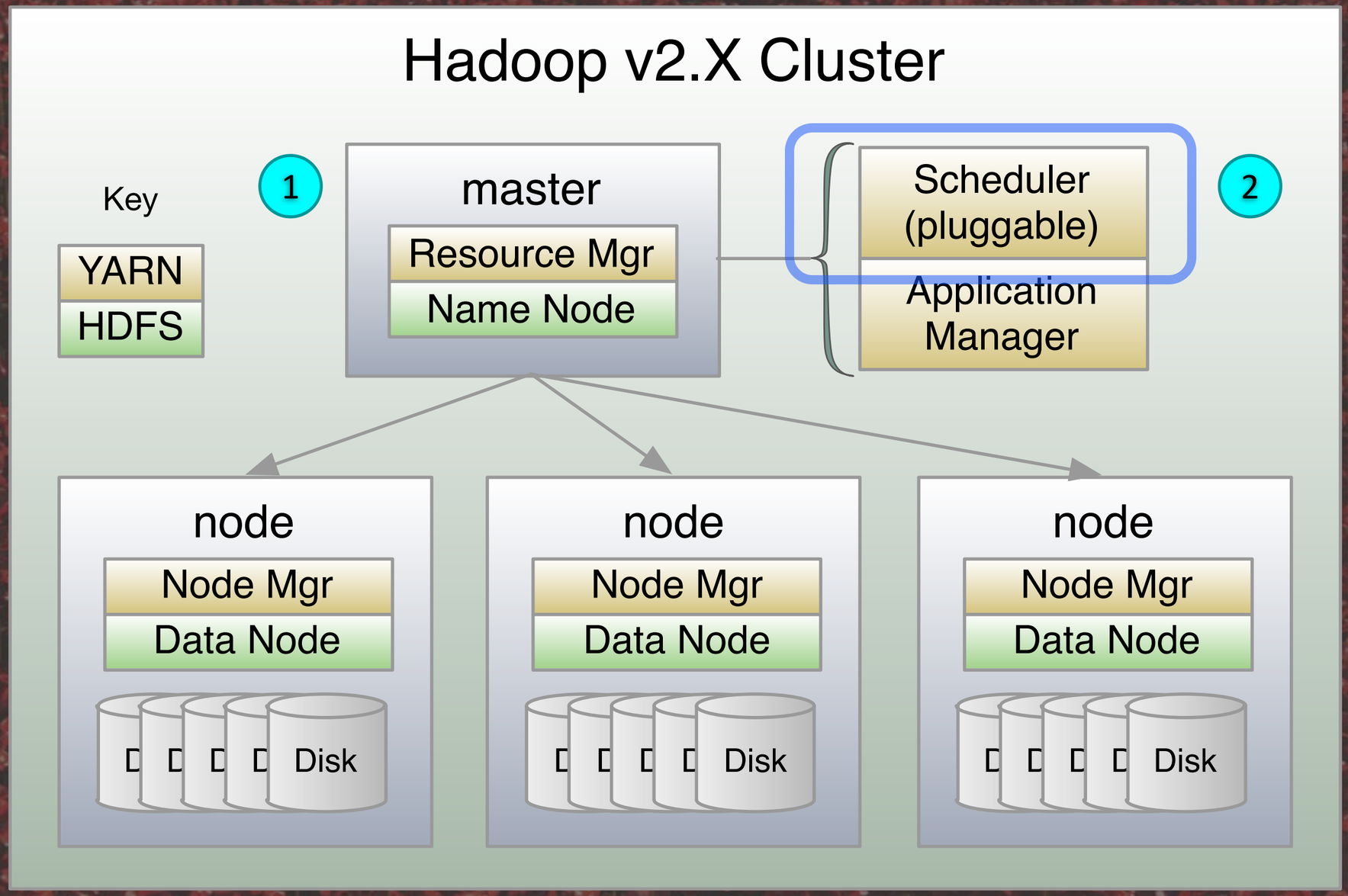
Resource Manager



Tuesday, October 20, 15

The RM accepts job submissions, performs security checks, etc. Then it's passed to the Scheduler.

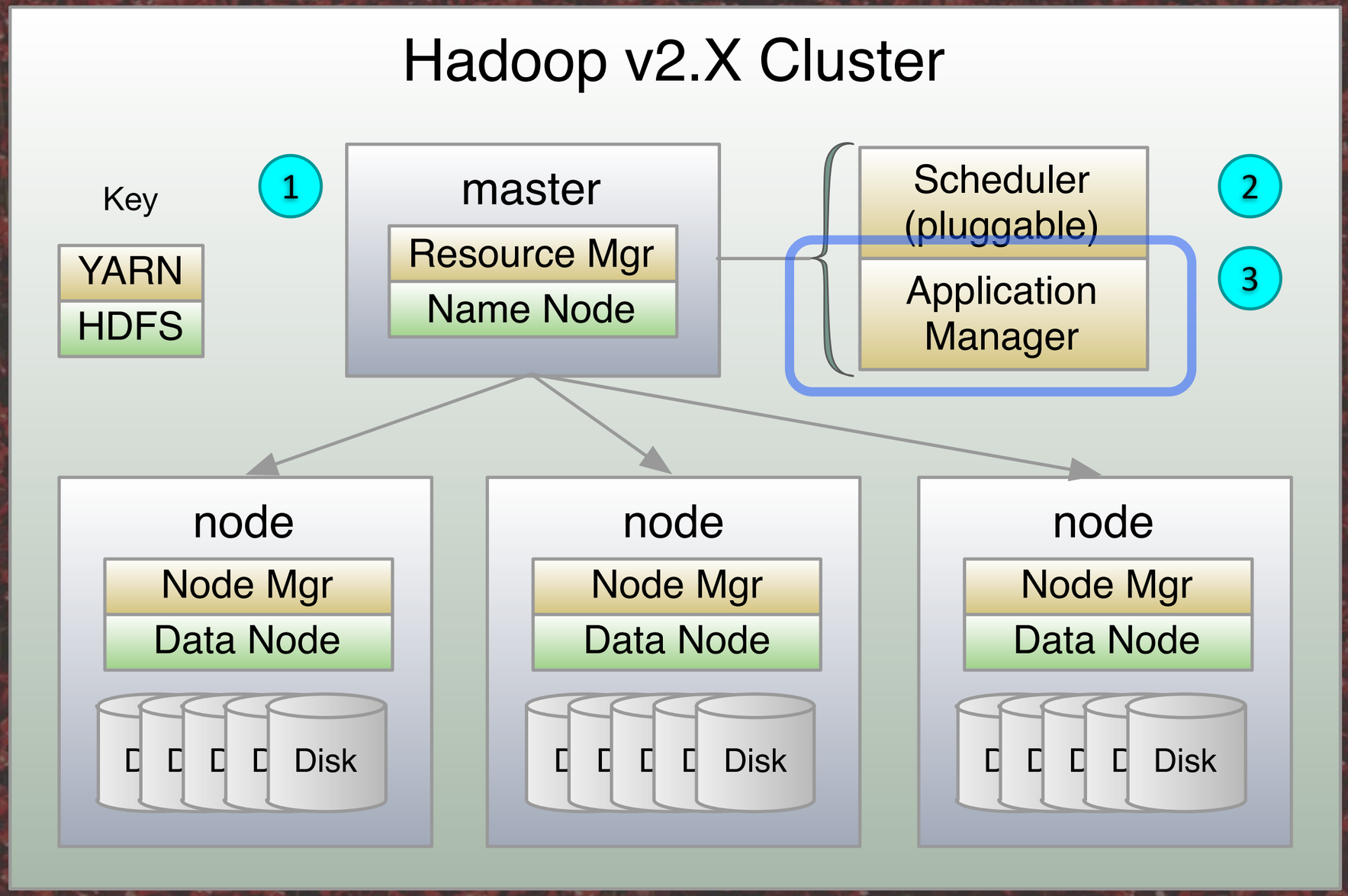
Scheduler



Tuesday, October 20, 15

The Scheduler moves the app from “accepted” to “running” once enough resources are available for it. This is a global scheduler, the familiar FIFO, Capacity, and Fair schedulers. Note that they are global choices. Some variations of resource negotiation are possible in the Application Masters (see below).
The cluster is treated as a resource pool. In fact, in a busy cluster, resources can be requested back from the applications.

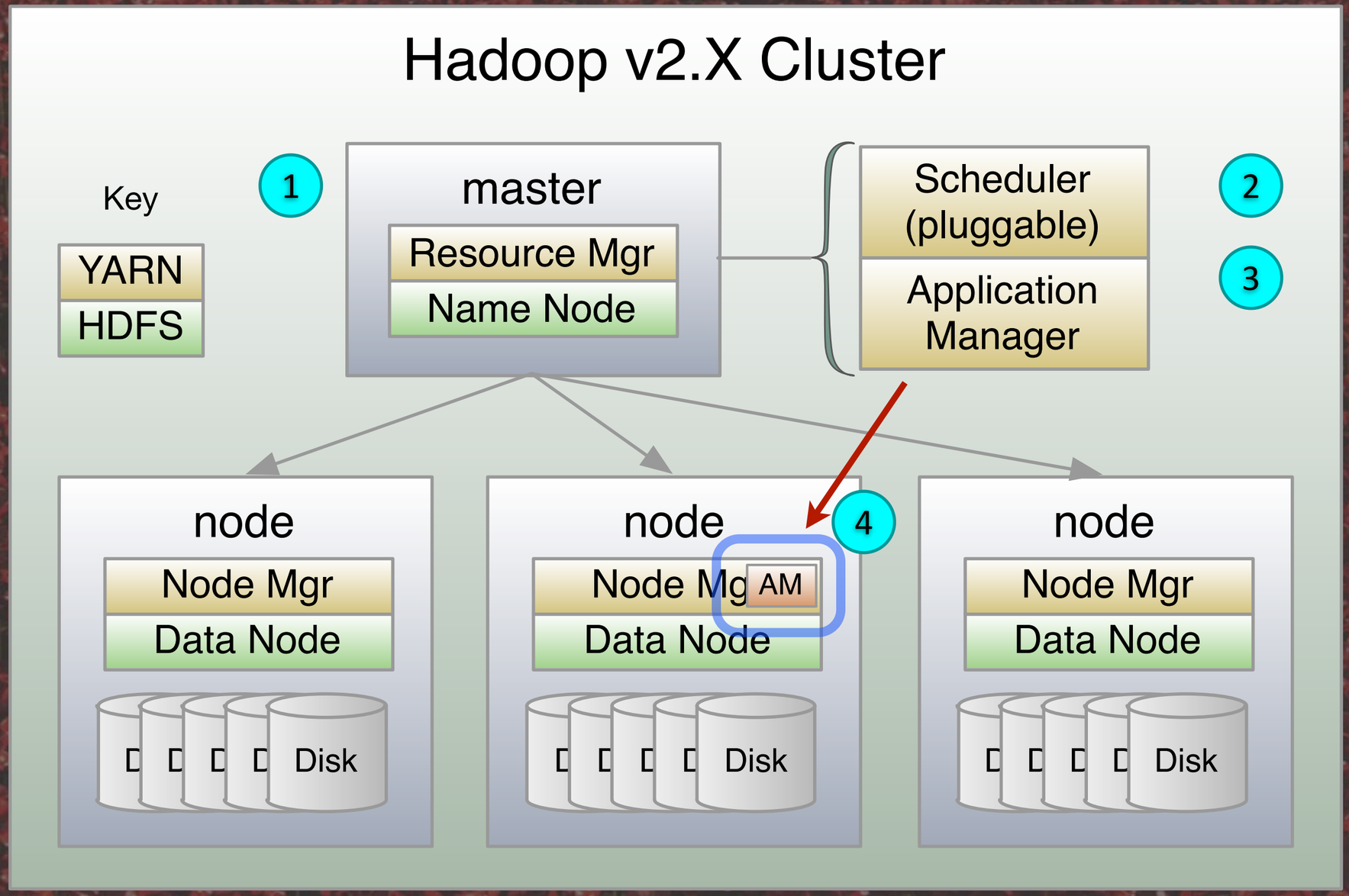
Application Manager



Tuesday, October 20, 15

The Application Manager manages user jobs.

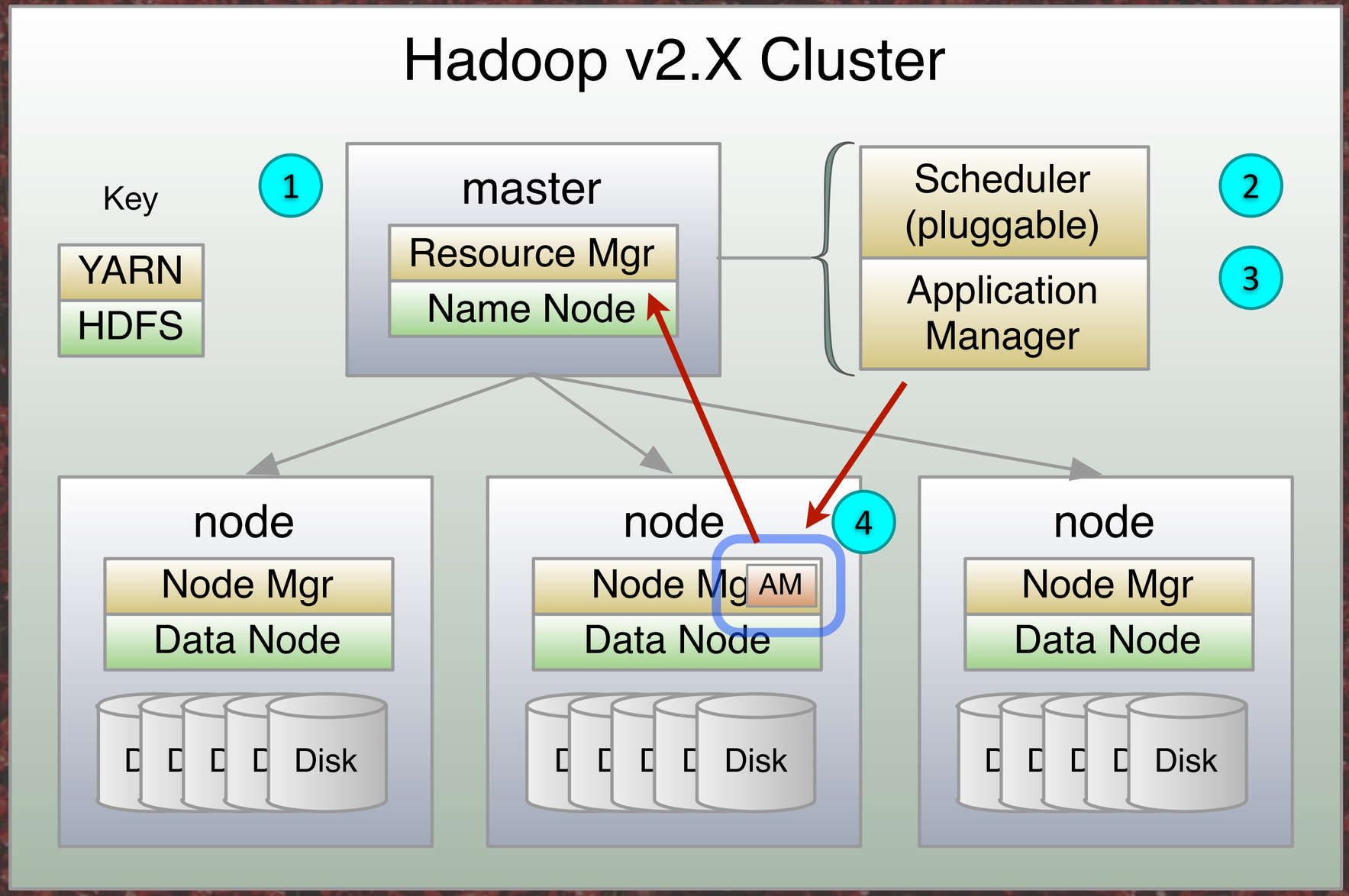
Application *Master*



Tuesday, October 20, 15

The Application Manager spawns one Application Master (AM) on one of the cluster nodes inside a container. This AM understands the compute engine, e.g., MapReduce, Spark.

More Resource Negotiation

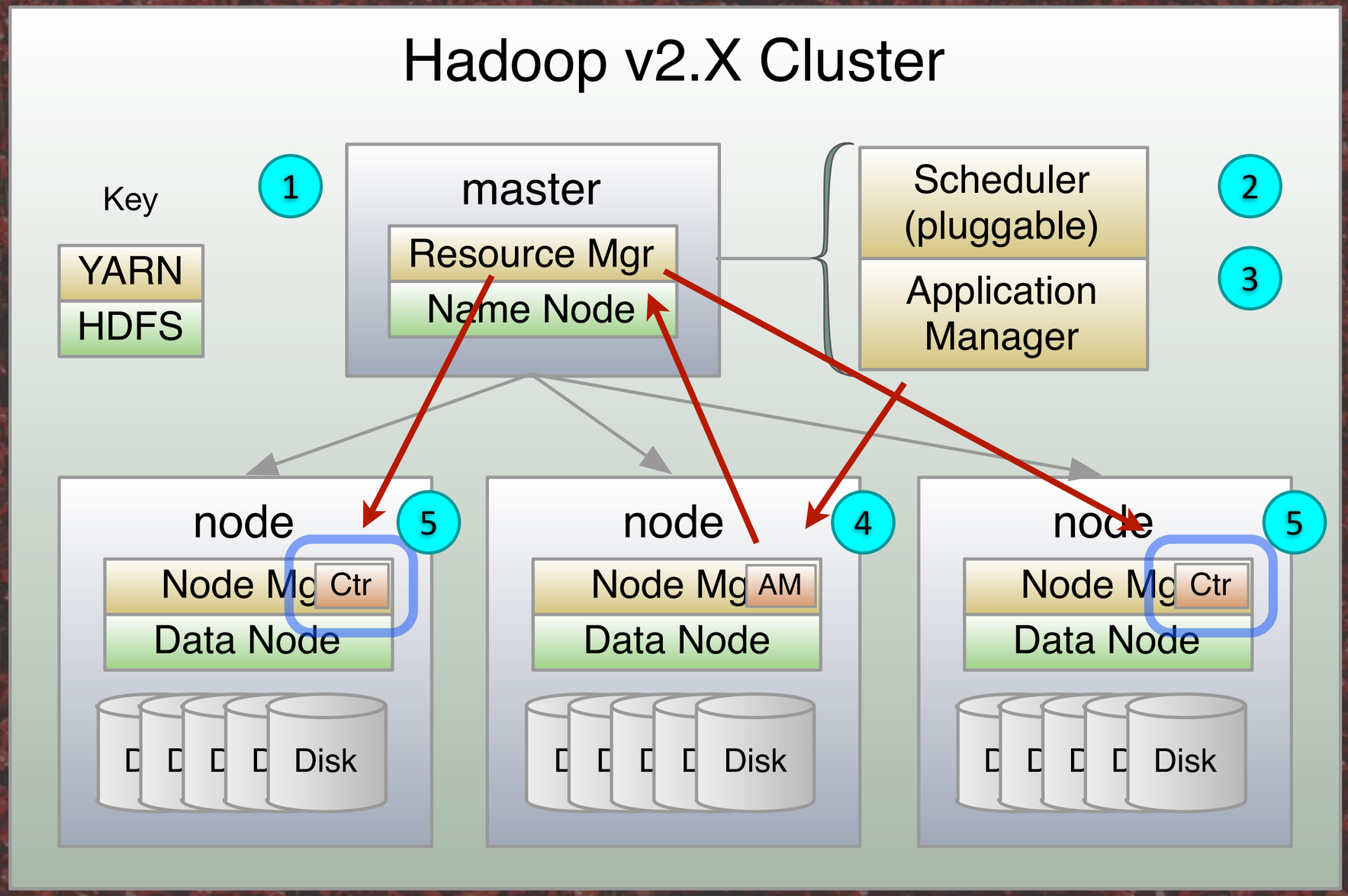


Tuesday, October 20, 15

The AM manages the job lifecycle, makes resource requests of the RM for resources on the nodes, sometimes with locality preferences (e.g., where HDFS blocks are located) and other container properties, dynamically scales resource consumption through containers, manages the flow of execution (e.g., MR job “stages”) handles failures, etc. When a resource is scheduled for the AM by the RM, a lease is created that the AM will pick up on the next heartbeat.

The AM communicates with user code in containers through Google Protocol Buffers, so it’s agnostic to the code language, etc. Also, each AM supports application-specific logic for features like data locality and other optimizations.

Containers

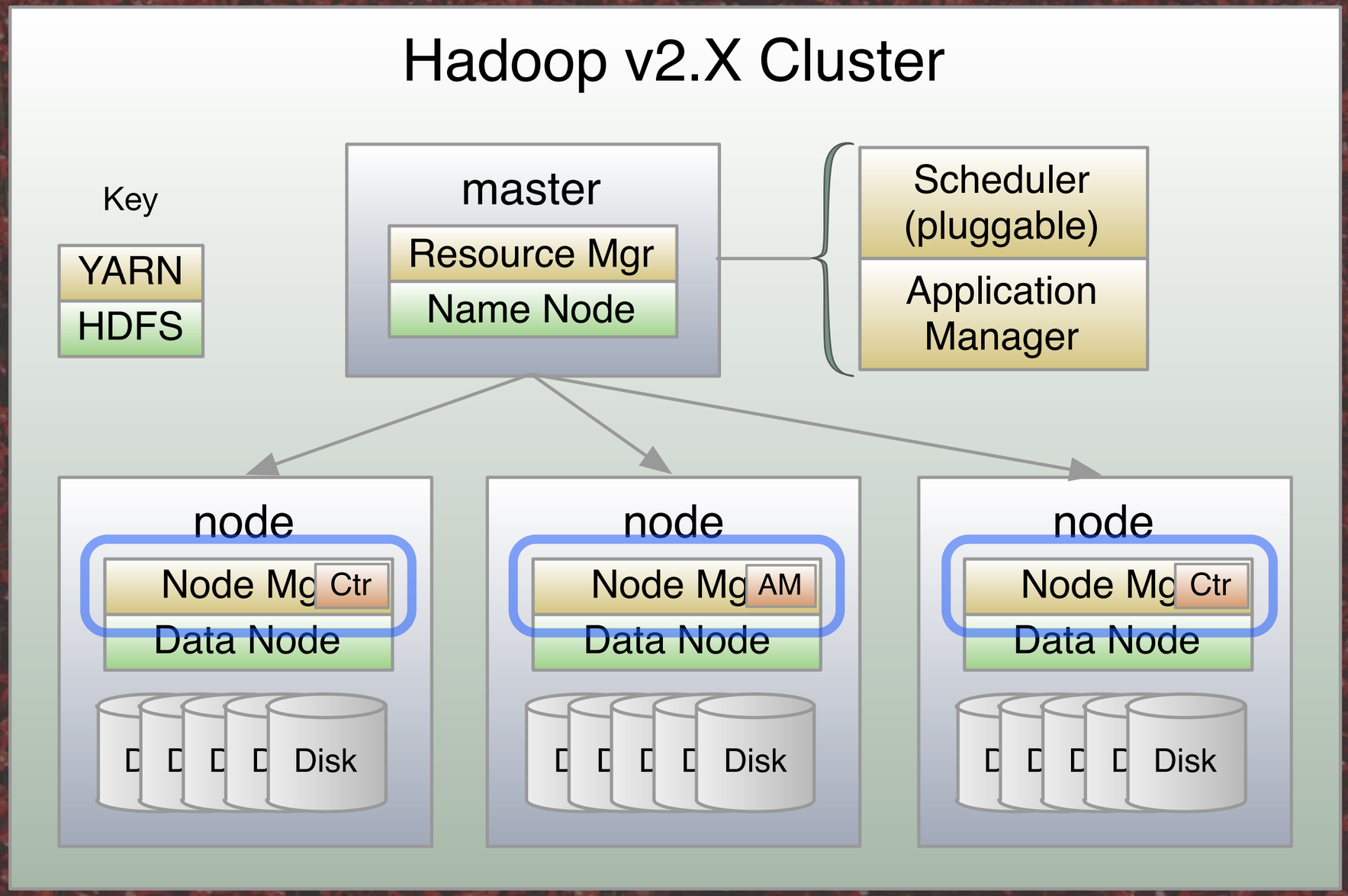


Tuesday, October 20, 15

Resources are scheduled in new containers on the nodes.

This is a form of “two-level scheduling” with responsibilities split between the master processes and the AMs.

Node Manager



Tuesday, October 20, 15

Finally, the containers are supervised locally by the Node Managers. They monitor resource usage by the containers, node health, communicate to the RM, etc.

A Few Key Points

- Resources are *dynamic*.
- Resources: CPU cores & memory.
- Scheduler is pluggable, but (mostly) global.
- Container model works best for "compute" jobs…

Tuesday, October 20, 15

We'll compare with Mesos. YARN doesn't yet manage disk space and network ports, but they are being considered. Scheduling is primarily a global concern and uses the Fair Scheduler, Capacity Scheduler, etc.

Today, HDFS isn't managed by YARN

Tuesday, October 20, 15

YARN's focus is on managing compute jobs and their constituent tasks, rather than providing comprehensive resource management of any kind of resource.

But greater flexibility is planned.

http://hortonworks.com/blog/evolving-apache-hadoop-yarn-provide-resource-workload-management-services/

Tuesday, October 20, 15

This greater flexibility is a so-called “container delegation” model that makes app container handling more flexible in ways that are necessary to support a wider variety of services, such as those for HDFS. Today you can’t use YARN to manage a Cassandra ring on the same cluster, for example.

Tuesday, October 20, 15

Something New

Mesos

mesos.apache.org

Tuesday, October 20, 15

Now let's discuss Mesos.

Mesos Cluster

Key

Mesos

Spark

HDFS

master

Mesos Master

master / client

Spark FW | Sched. | Job 1

master / client

HDFS FW | Sched. | Job 1

node

Mesos Slave

Name Node Executor

task1 | …

Spark Executor

task1 | …

node

Mesos Slave

Data Node Executor

… | …

Spark Executor

… | …

Disk

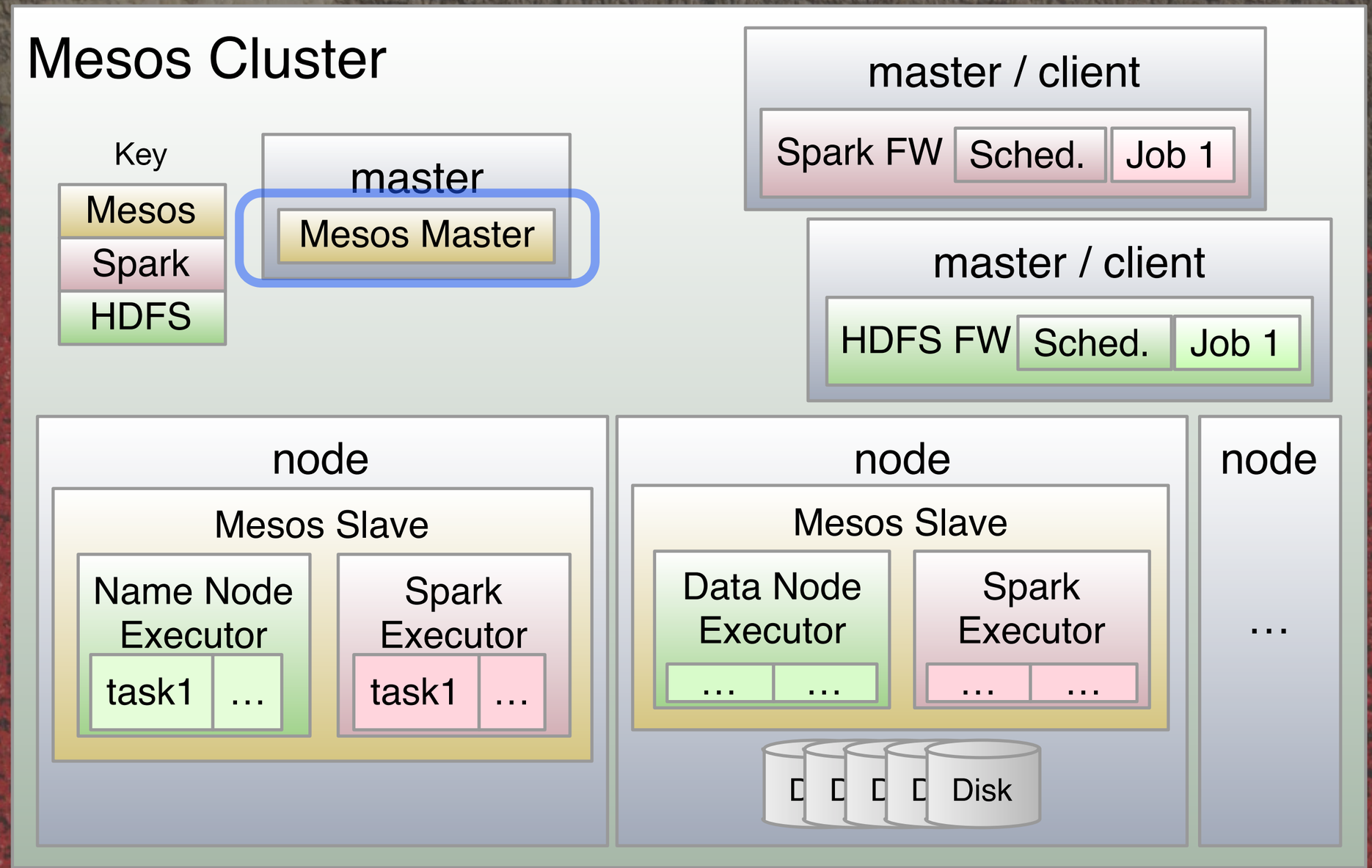
node

…

Tuesday, October 20, 15

Schematic view of a Mesos cluster that will run Spark and HDFS.

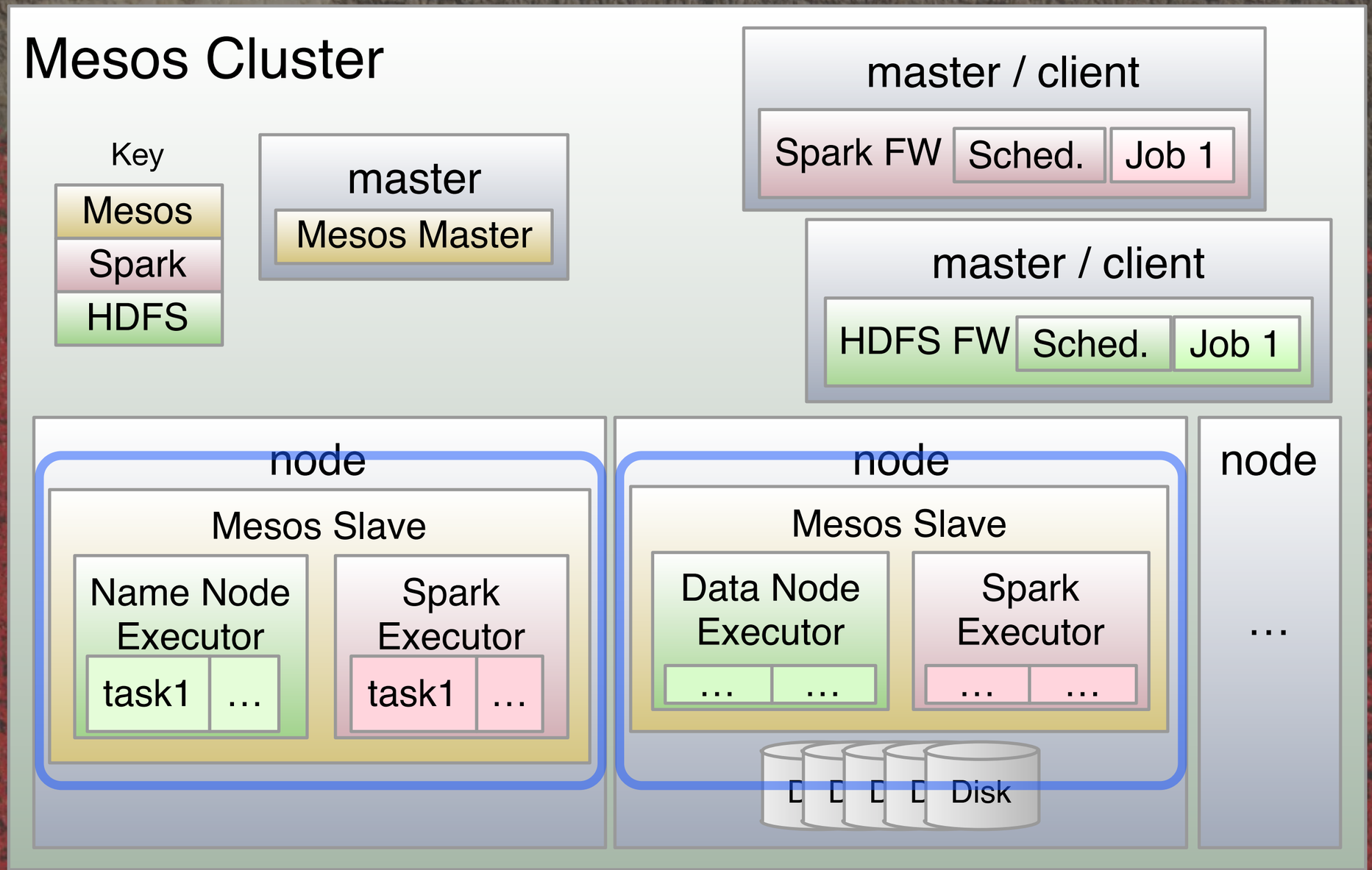
Mesos Master



Tuesday, October 20, 15

Like the YARN Resource Manager. Mesos Master failover to standbys can be managed by Zookeeper (not shown).

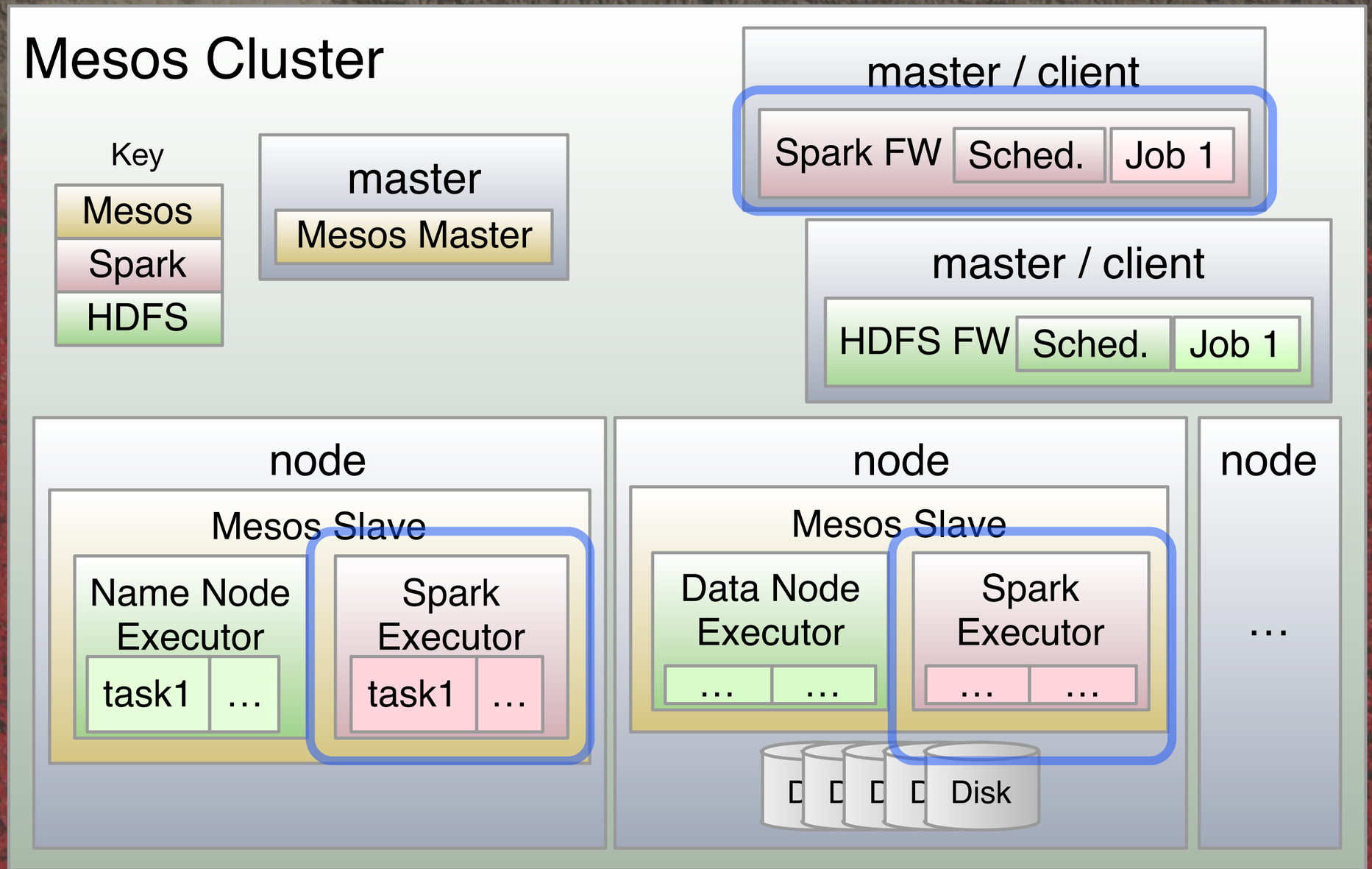
Mesos Slaves



Tuesday, October 20, 15

The Master controls a Slave daemon on each node, which controls the application Executors running on that node. (The Mesos community is migrating to the term “worker” instead of the more loaded term “slave”.)

Frameworks



Tuesday, October 20, 15

Each application is a Framework (FW), with a Scheduler. Normally, this will run on the cluster somewhere, not necessarily the the master node, but it could run on some “client” node. It communicates with the Master (using a Scheduler Driver) to negotiate scheduling and resources, etc. Each node has an Executor under which all processes for the Framework are executed. Here I’ve circled just those for Spark.

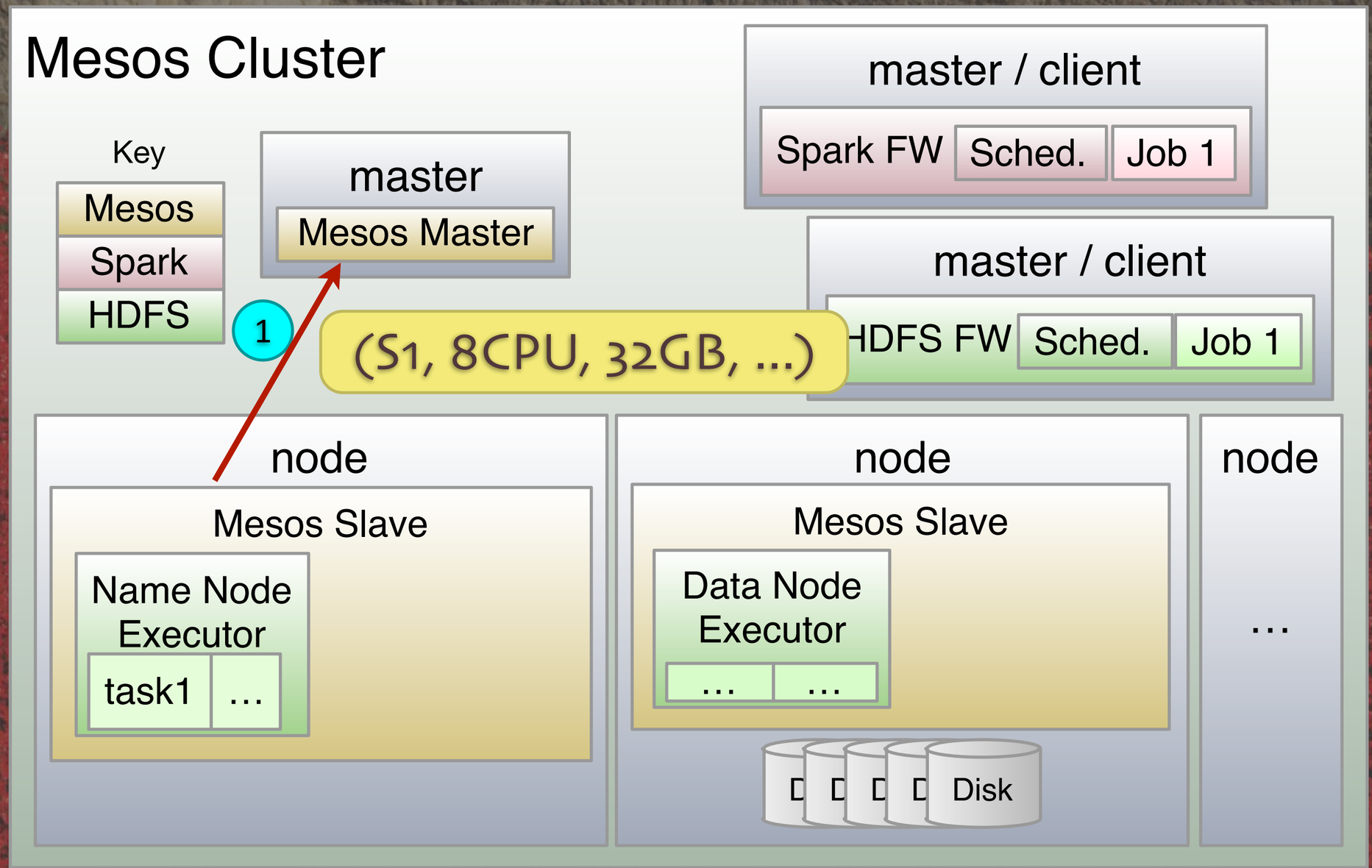
A framework could manage many “jobs”, such as submitted Spark batch jobs, as we’ll see.

For HDFS, note how the master processes, Name Node and Data Node are run inside executors.

Tuesday, October 20, 15

A key strategy in Spark is to offer resources to frameworks, which can chose to accept or reject them. Why reject them? The offer may not be sufficient for the need, but it's also a technique for informally imposing policies of interest to the application, such as data locality.

Example: Spark

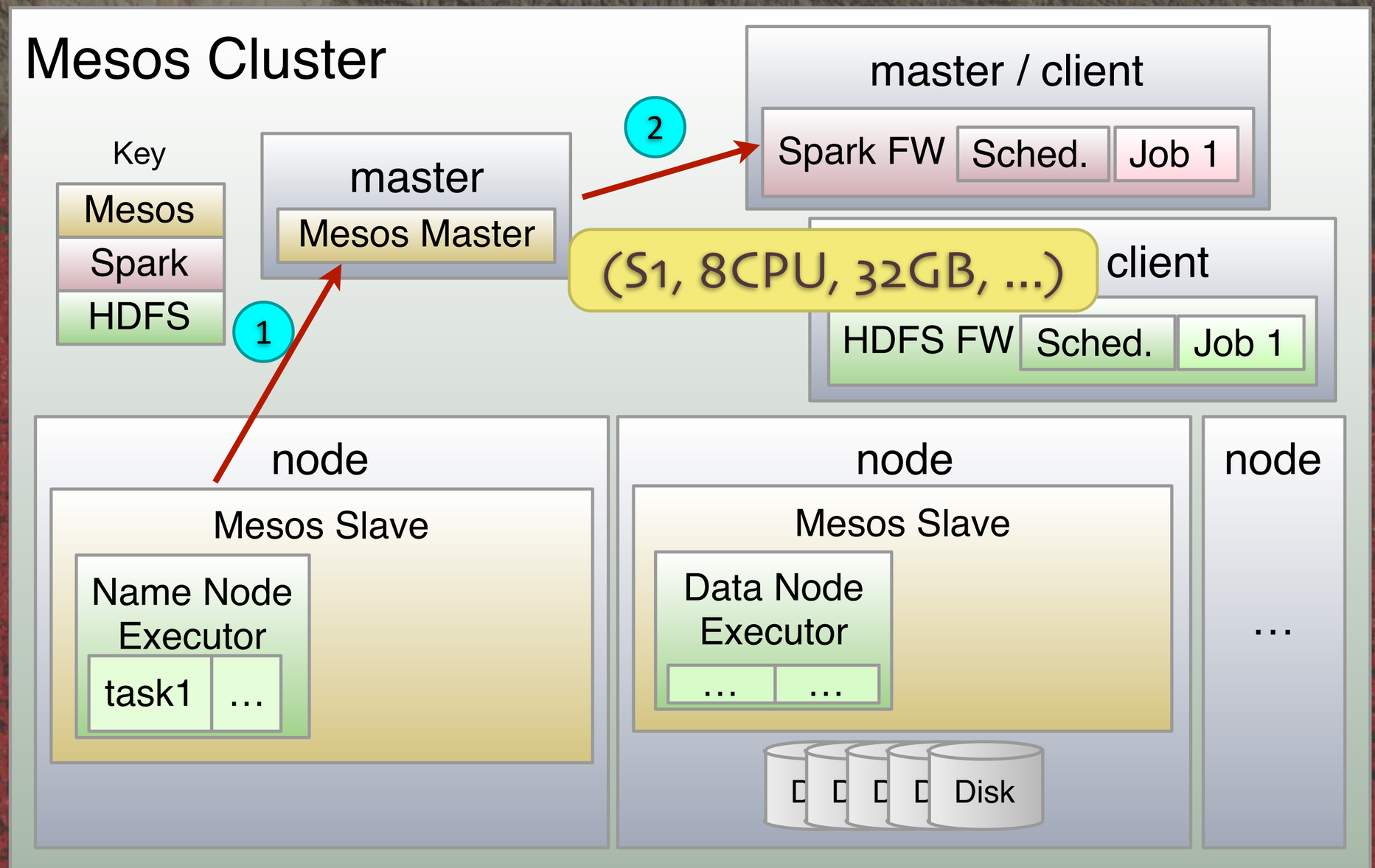


Tuesday, October 20, 15

Assume that HDFS is already running and we want to allocate resources and start executors running for Spark.
1. A slave (#1) tells the Master (actually the Allocation policy module embedded within it) that it has 8 CPUs, 32GB Memory. (Mesos can also manage ports and disk space.)

Adapted from http://mesos.apache.org/documentation/latest/mesos-architecture/

Example

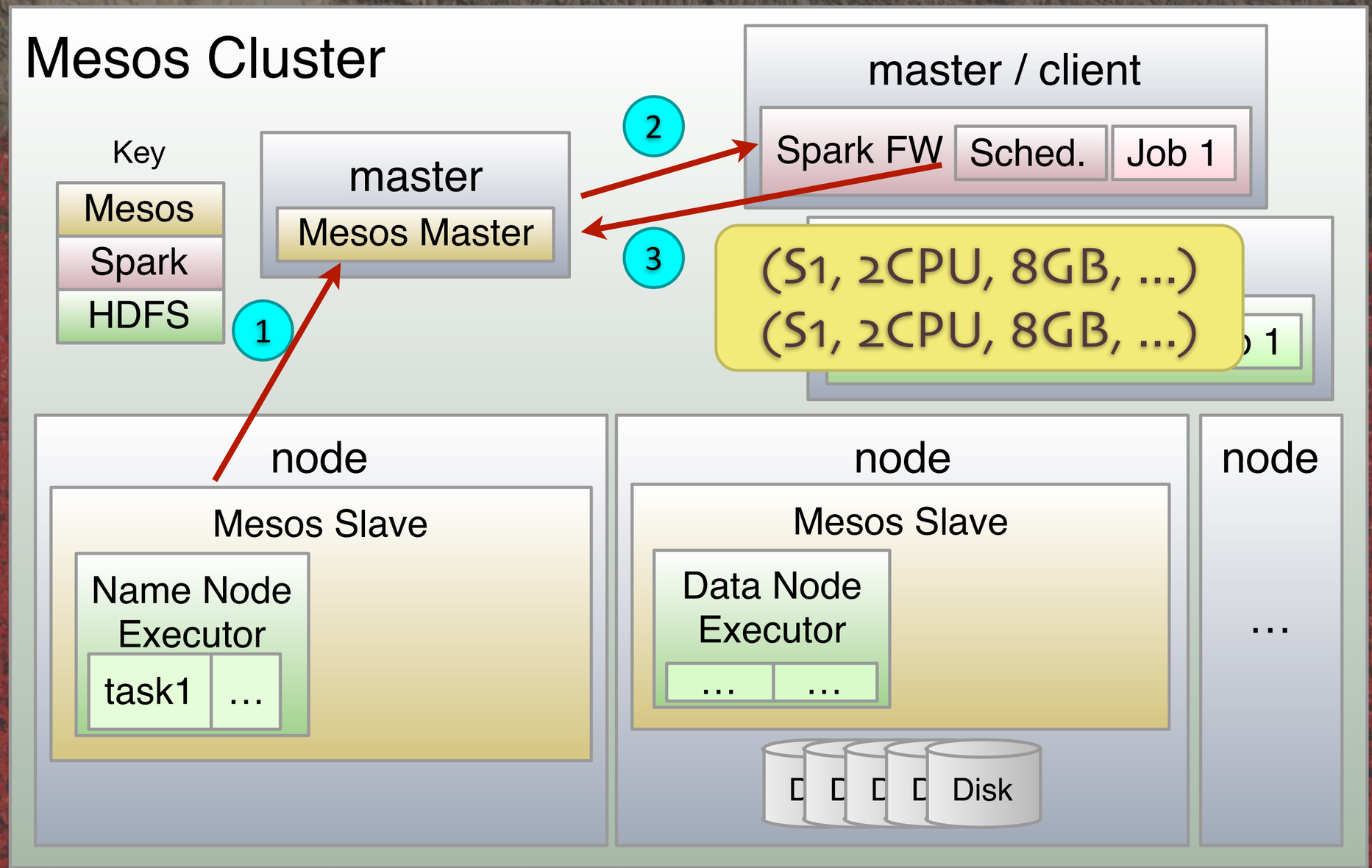


Tuesday, October 20, 15

A resource offer example. Adapted form http://mesos.apache.org/documentation/latest/mesos-architecture/

2. The Allocation module in the Master decides that all the resources should be offered to the Spark Framework.

Example

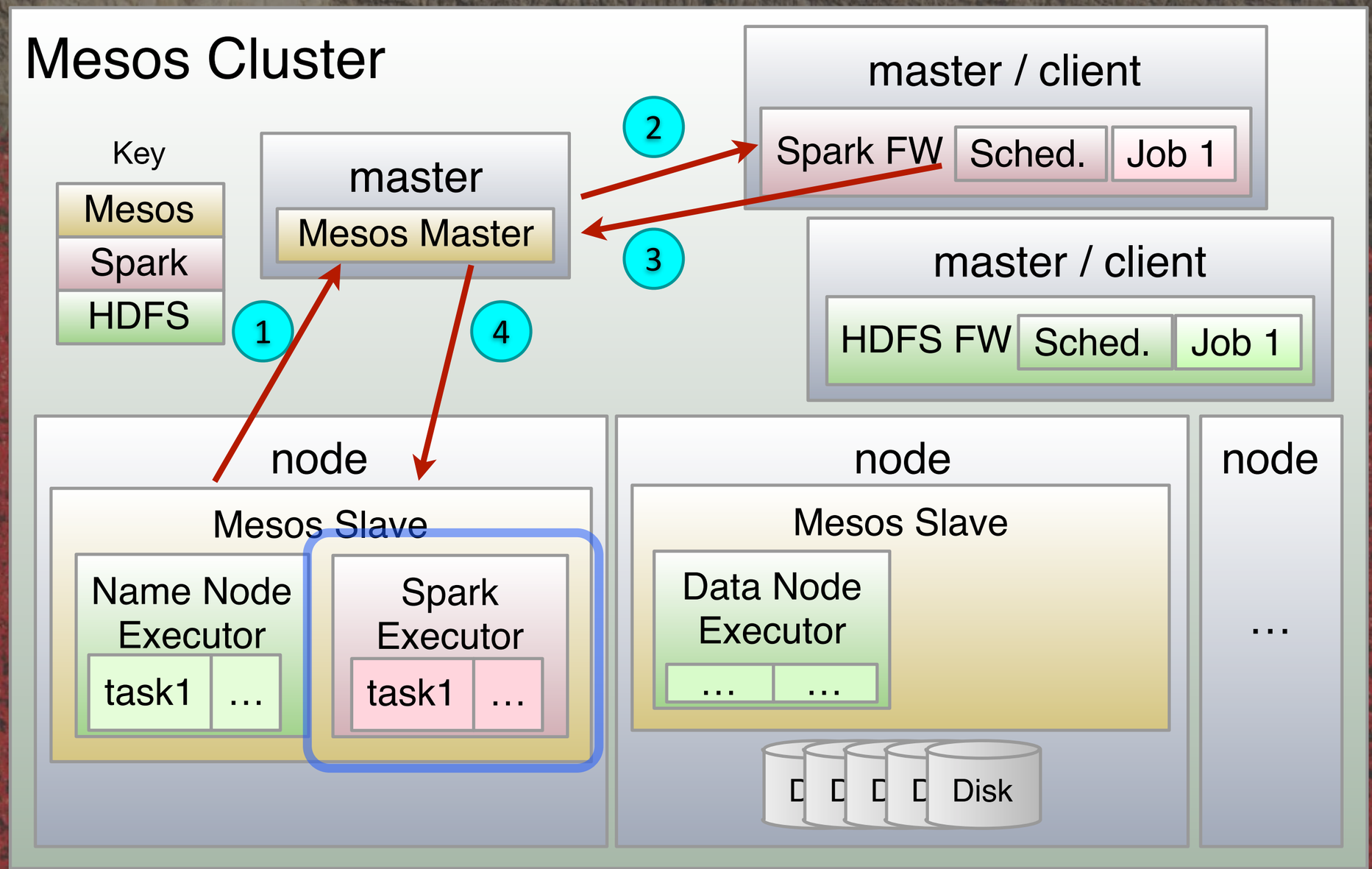


Tuesday, October 20, 15

A resource offer example. Adapted form http://mesos.apache.org/documentation/latest/mesos-architecture/

3. The Spark Framework Scheduler replies to the Master to run two tasks on the node, each with 2 CPU cores and 8GB of memory. The Master can then offer the rest of the resources to other Frameworks.

Example



Tuesday, October 20, 15

A resource offer example. Adapted form http://mesos.apache.org/documentation/latest/mesos-architecture/

4. The master spawns the executor (if not already running - we'll dive into this bubble!!) and the subordinate tasks.

A Few Key Points

- Resources are *dynamic*.
- Resources: CPU cores, memory, *plus* disk, ports.
- Scheduling, resource negotiation fine grained and per-framework.

http://mesos.berkeley.edu/mesos_tech_report.pdf

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

Benjamin Hindman, Andy Konwinski, Matei Zaharia,
Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica
University of California, Berkeley

Thursday 30th September, 2010, 12:57

Abstract

We present Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data replication. Mesos shares resources in a fine-grained manner, allowing frameworks to achieve data locality by taking turns reading data stored on each machine. To

The solutions of choice to share a cluster today are either to statically partition the cluster and run one framework per partition, or allocate a set of VMs to each framework. Unfortunately, these solutions achieve neither high utilization nor efficient data sharing. The main problem is the mismatch between the allocation granularities of these solutions and of existing frameworks. Many frameworks, such as Hadoop and Dryad, employ a fine-

Tuesday, October 20, 15

See this research paper by Benjamin Hindman, the creator of Mesos, and Matei Zaharia, the creator of Spark for more details.

http://mesos.berkeley.edu/mesos_tech_report.pdf

"Our results show that Mesos can achieve near-optimal data locality when sharing the cluster among diverse frameworks, can scale to 50,000 (emulated) nodes, and is resilient to failures."

Tuesday, October 20, 15

See this research paper by Benjamin Hindman, the creator of Mesos, and Matei Zaharia, the creator of Spark for more details.

http://mesos.berkeley.edu/mesos_tech_report.pdf

*"To validate our hypothesis ..., we have also built a new framework on top of Mesos called **Spark**..."*

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center

Benjamin Hindman, Andy Konwinski, Matei Zaharia,

University of California, Berkeley

Abstract

We present Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data replication. Mesos shares resources in a fine-grained manner, allowing frameworks to achieve data locality by taking turns reading data stored on each machine. To

The solutions of choice to share a cluster today are either to statically partition the cluster and run one framework per partition, or allocate a set of VMs to each framework. Unfortunately, these solutions achieve neither high utilization nor efficient data sharing. The main problem is the mismatch between the allocation granularities of these solutions and of existing frameworks. Many frameworks, such as Hadoop and Dryad, employ a fine-

Tuesday, October 20, 15

See this research paper by Benjamin Hindman, the creator of Mesos, and Matei Zaharia, the creator of Spark for more details. (Bold highlight not in the original paper ;)

Containers

- Mesos Containerizer:
  - Lightweight, Linux cgroups, ...
  - Can compose with extra file, disk, PID isolators.

Tuesday, October 20, 15

The MesosContainerizer is very lightweight, because by default it doesn't actually isolate anything by default, just report utilization. It can be composed with isolators for controlling the view of shared filesystems, disks, and pid namespaces. While limiting protection from rogue tasks, it imposes minimal overhead for performance critical apps.

Containers

- Docker Containerizer:
  - Launch docker images as tasks or executors.
- More complete isolation.

Tuesday, October 20, 15

Use Docker.

Containers

- External Containerizer:
  - Protocol for custom containerizers.

Tuesday, October 20, 15

Use your own container system!

Apps on Mesos

- Apple's Siri
- MySQL - Mysos
- Cassandra
- HDFS
- YARN! - Myriad
- others...

Tuesday, October 20, 15

Mesos' flexibility has made it possible for many frameworks to be supported on top of it. For more examples, see http://mesos.apache.org/documentation/latest/mesos-frameworks/ Myriad is very interesting as a bridge technology, allowing (once it's mature) legacy YARN-based apps to enjoy the flexible benefits of Mesos. More on this later...

Tuesday, October 20, 15

Tuesday, October 20, 15

What's the "elevator pitch" for Mesos vs. YARN? The details are changing constantly.

YARN

- ☑ Mature.
- ☑ Large developer community.
- ☑ Large ecosystem of 3rd-party apps.
- ☑ Supported by Hadoop vendors.
- ☑ Improving support for long-running services.

Tuesday, October 20, 15

The last bullet is about dynamic resource allocation/de-allocation.

YARN

- Limits of scheduler & container models:
  - Limited to “~compute” jobs.
  - Hadoop, Big Data-centric.
  - Container models: Docker, etc.
  - Other resources: disk, network, ...?

Mesos

- ☑ Next generation resource model.
  - ☑ More resource “kinds”.
  - ☑ Flexible containerization.
  - ☑ Wider class of applications, even all of Hadoop!
  - ☑ More fine-grained scheduling.

Tuesday, October 20, 15

Kinds of resources being added include disk space, network bandwidth, in addition to the familiar CPU cores and memory. You can run your entire infrastructure, even Hadoop on top of Mesos.

Mesos

☑ Easier programming model for 3rd-party apps.

Tuesday, October 20, 15

YARN is hard to program against. Mesos is easier on developers.

Mesos

- Smaller, less mature ecosystem.
  - Doesn't fill all Enterprise "niches".
- Smaller developer community.
- Fewer support vendors.

Tuesday, October 20, 15

While Mesos is actually older than YARN, most of it's growth happened rapidly in the last several years at Twitter, Airbnb, and Mesosphere. Third-party Mesos tools are also relatively new. The "niches" include mature tools for security, data governance, regulatory compliance, etc. Mesosphere is the only vendor offering commercial support for Mesos.

Spark on Mesos

spark.apache.org/docs/latest/running-on-mesos.html

Both Born at UC Berkeley

- Benjamin Hindman & Matei Zaharia
- ~2007-2009
- Both now top-level Apache projects

Tuesday, October 20, 15

A little more background

Spark Is Cluster Agnostic

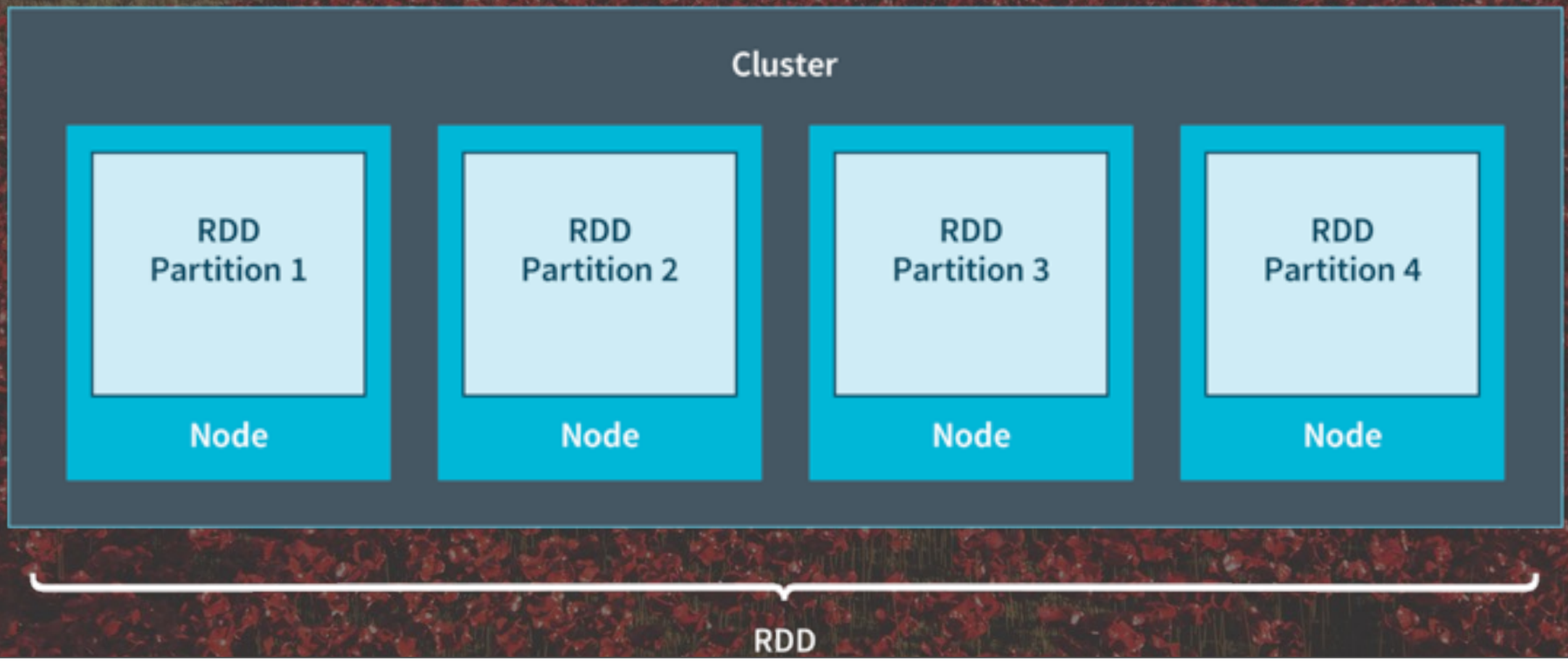
- YARN
- Mesos
- Standalone mode
- Local (1 machine)
- Cassandra, Riak, etc. clusters.

Tuesday, October 20, 15

Spark has a protocol for cluster management and several options are implemented. As the Mesos research paper hinted, Benjamin Hindman and Matei Zaharia were grad students together at UC Berkeley. Spark and Mesos were developed about the same time and Mesos was the first cluster implementation (maybe after standalone mode).
Local mode is a HUGE advantage of MapReduce, because it makes it easy to test the logic of your program just like any other JVM app. If your “production” data is small enough, you could use local mode for “real” work, too. DB vendors like Cassandra and Basho are integrating Spark.

Compute Model

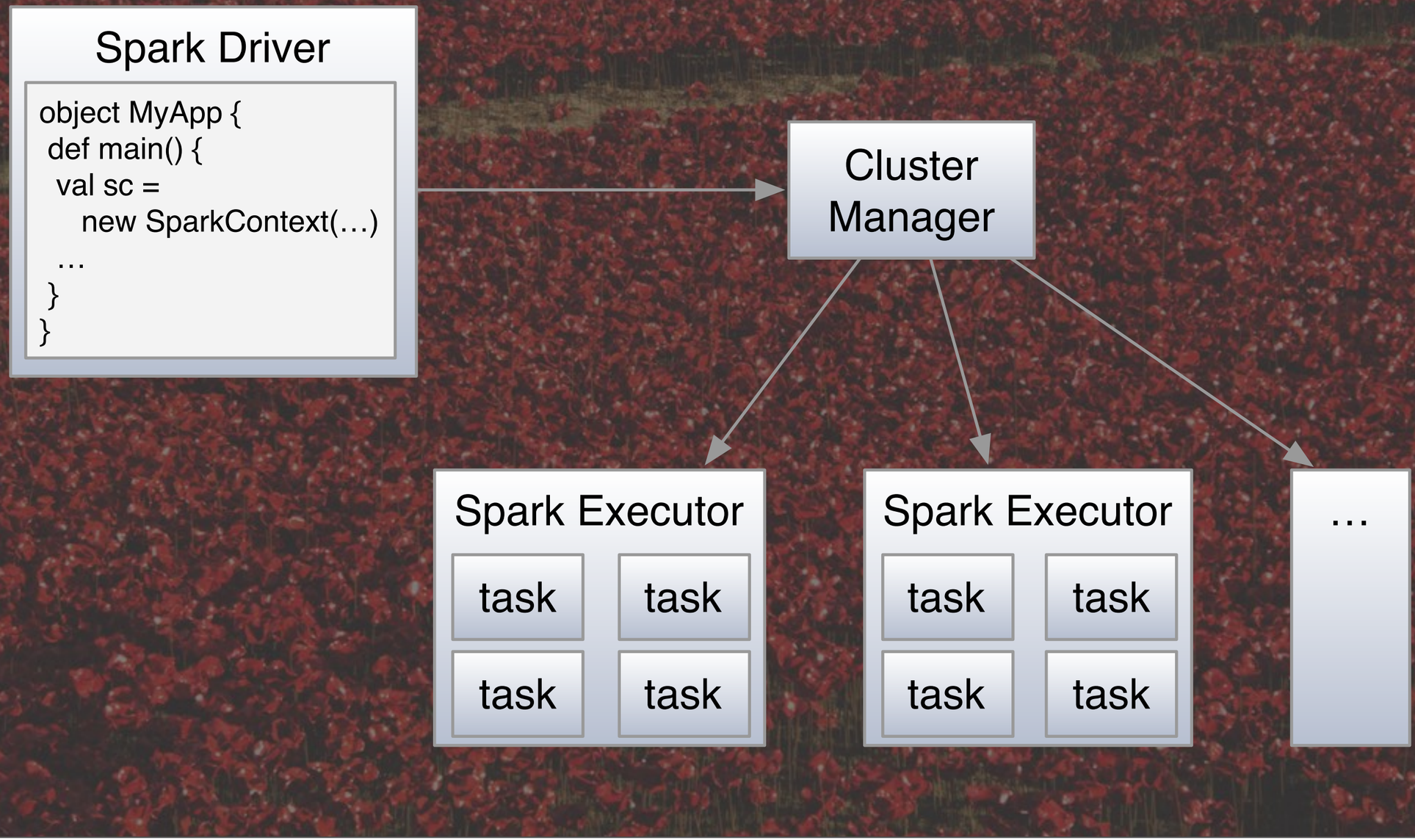
- RDDs: Resilient Distributed Datasets



Tuesday, October 20, 15

Your data is partitioned across the cluster. The overall data structure behaves like a collection with functional/SQL operations on it. It's resilient because if one partition is lost (say the node crashes), Spark can reconstruct it from earlier RDDs in the processing pipeline, all the way back to the input.

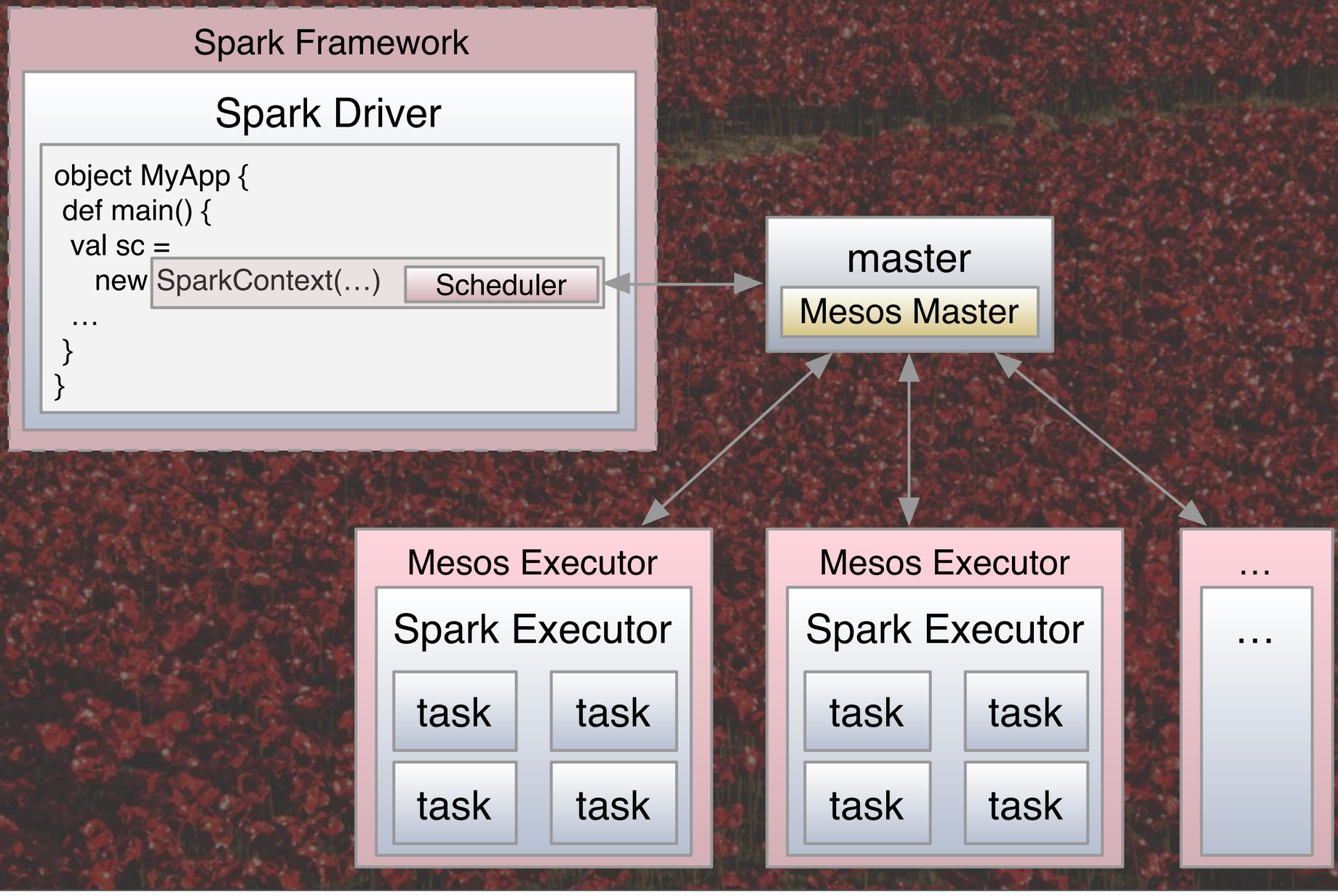
Cluster Abstraction



Tuesday, October 20, 15

For Spark Standalone, the Cluster Manager is the Spark Master process. For Mesos, it's the Mesos Master. For YARN, it's the Resource Manager.

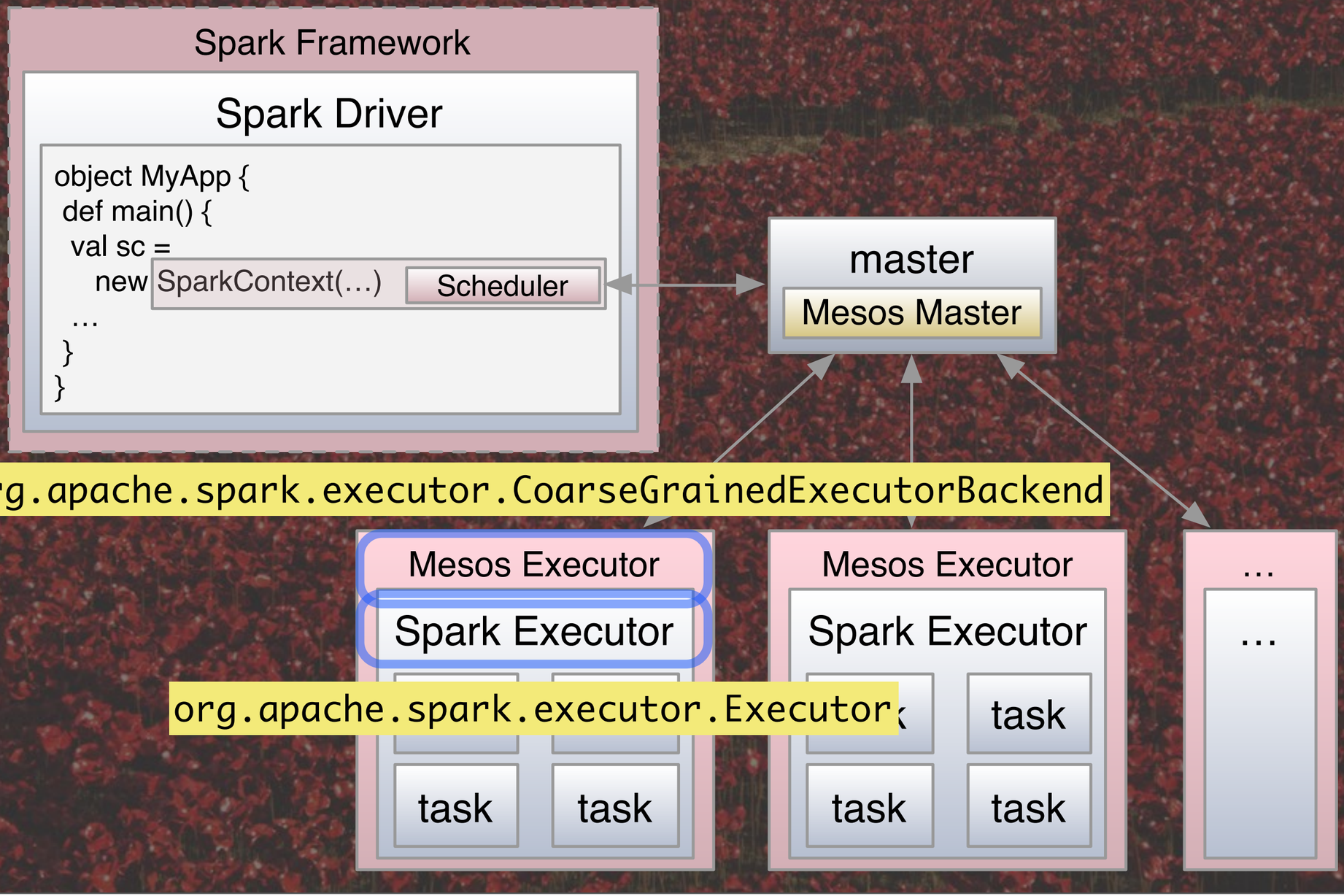
Mesos Coarse Grained Mode



Tuesday, October 20, 15

Unfortunately, because Spark and Mesos “grew up together”, each uses the same terms for concepts that have diverged. The “Mesos Executors” shown were called “Spark Executors” in previous slides. The actual Scala class name is org.apache.spark.executor.CoarseGrainedExecutorBackend. It has a “main” and a process It encapsulates a cluster-agnostic instance of Scala class org.apache.spark.executor.Executor, which manages the Spark tasks. for the nested “Spark Executor” is called CoarseGrainedExecutorBackend.
Note that the Spark “Framework” is an abstraction, not an actual service or process. Those are the constituents, like the scheduler, executors, etc.

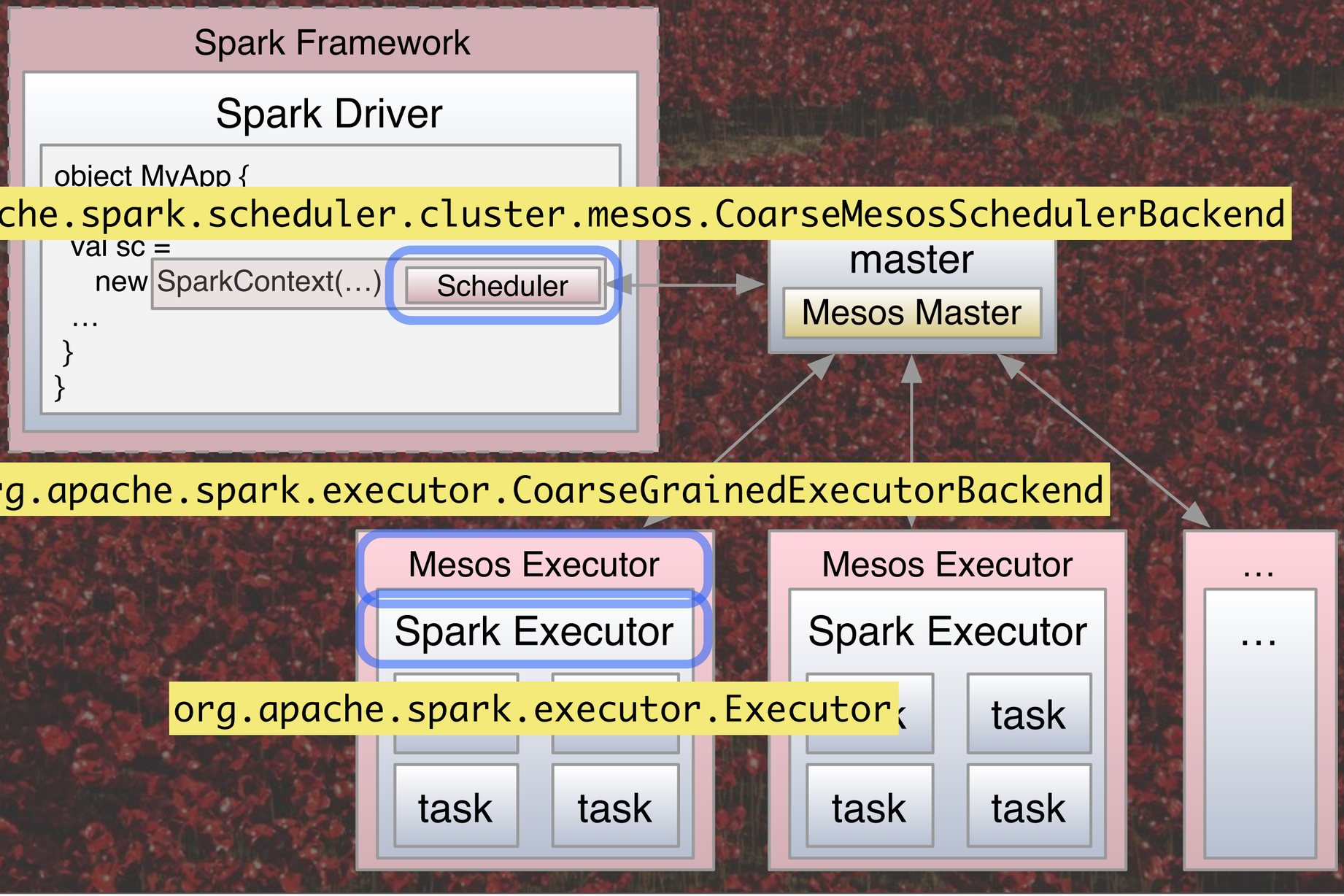
Mesos Coarse Grained Mode



Tuesday, October 20, 15

Unfortunately, because Spark and Mesos “grew up together”, each uses the same terms for concepts that have diverged. The “Mesos Executors” shown were called “Spark Executors” in previous slides. The actual Scala class name is org.apache.spark.executor.CoarseGrainedExecutorBackend. It has a “main” and a process It encapsulates a cluster-agnostic instance of Scala class org.apache.spark.executor.Executor, which manages the Spark tasks. Note that both are actually Mesos agnostic...

Mesos Coarse Grained Mode

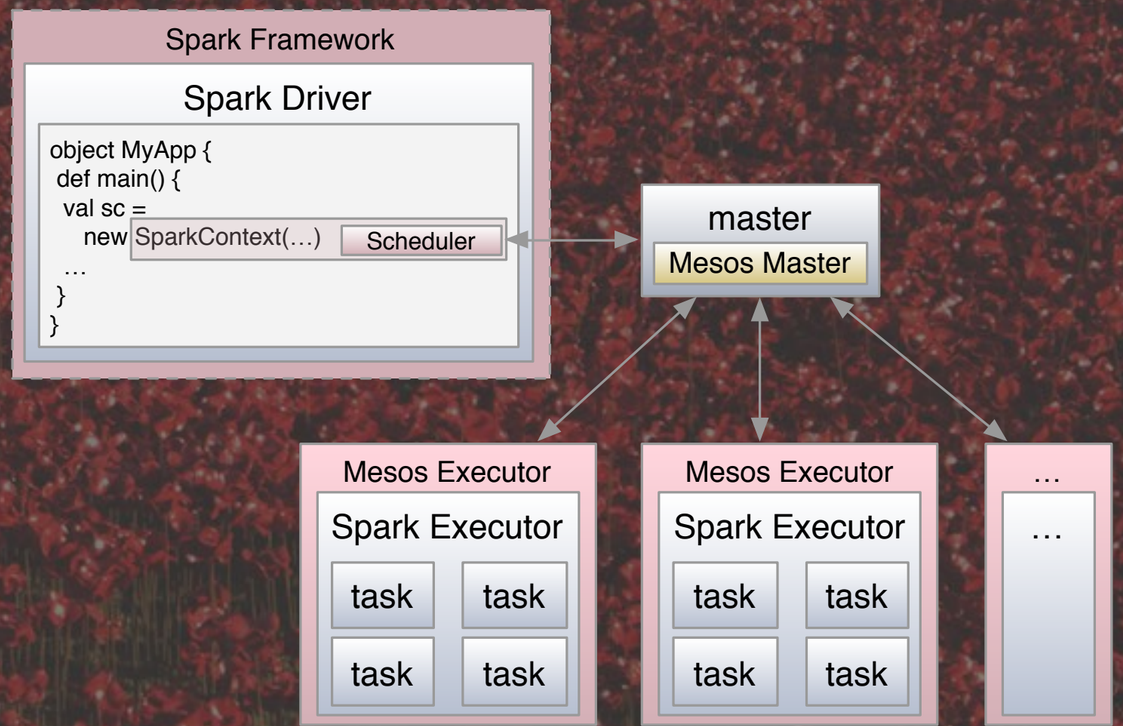


Tuesday, October 20, 15

The CoarseGrainedExecutorBackend process is started by the CoarseMesosSchedulerBackend, one per slave. So the coarse-grained executor is Mesos agnostic, but not the scheduler. One CoarseMesosSchedulerBackend instance is created by the SparkContext as a field in the instance.

Mesos Coarse Grained Mode

- One Mesos and one Spark executor for the job's lifetime.
- Tasks are spawned by Spark itself.



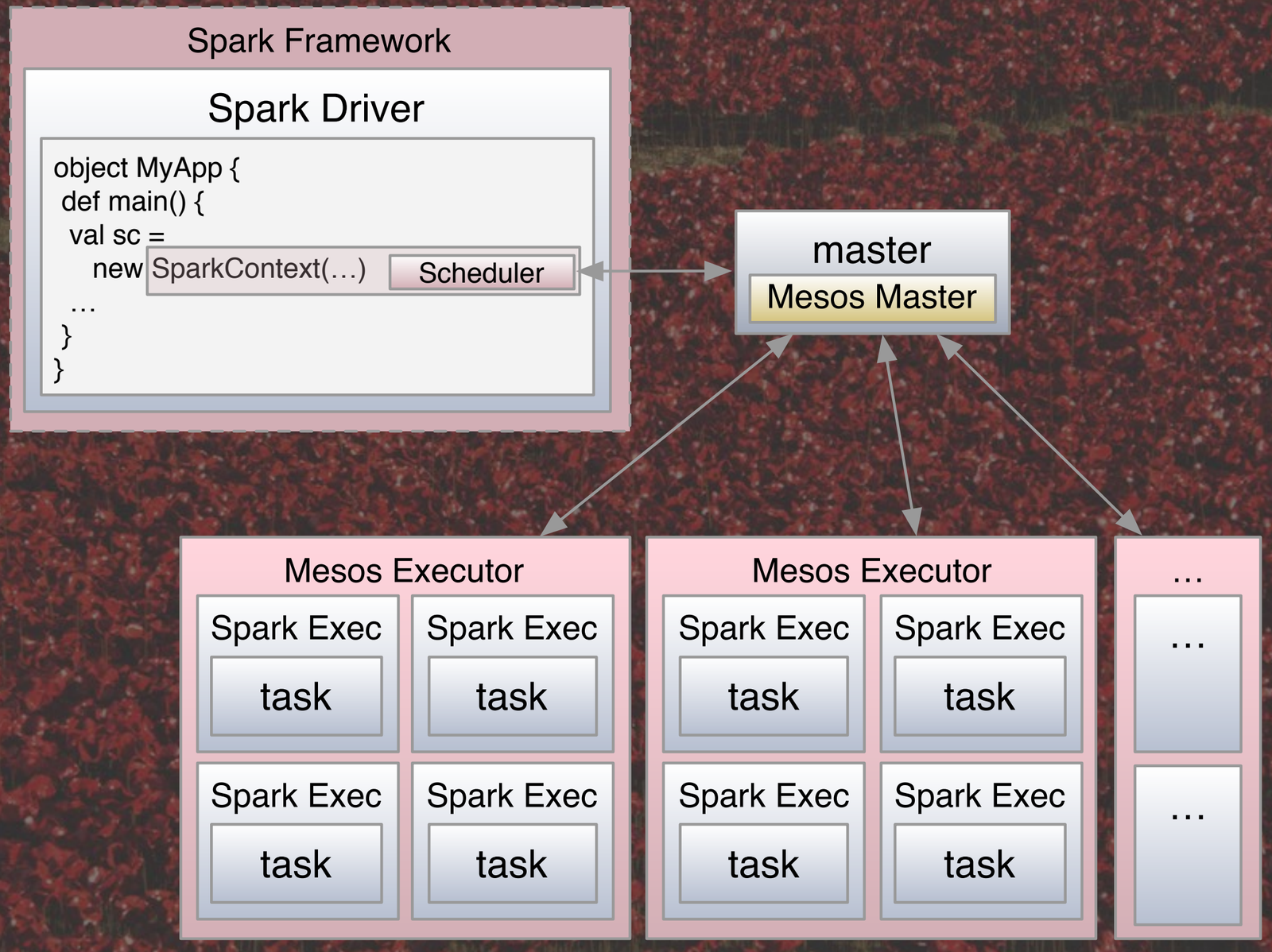
Mesos Coarse Grained Mode

- ☑ Fast startup for tasks:
  Better for interactive sessions.
- ☐ Resources locked up in larger Mesos task.
- But dynamic allocation!
- ☐ One task per slave!

Tuesday, October 20, 15

Dynamic allocation allows Mesos to reclaim unused resources by killing tasks. This feature is being enhanced to support over allocation, for cases where frameworks don’t actually use the full resources they’ve been allocated, but if the speculative oversubscription is wrong, the lower-priority tasks are killed.

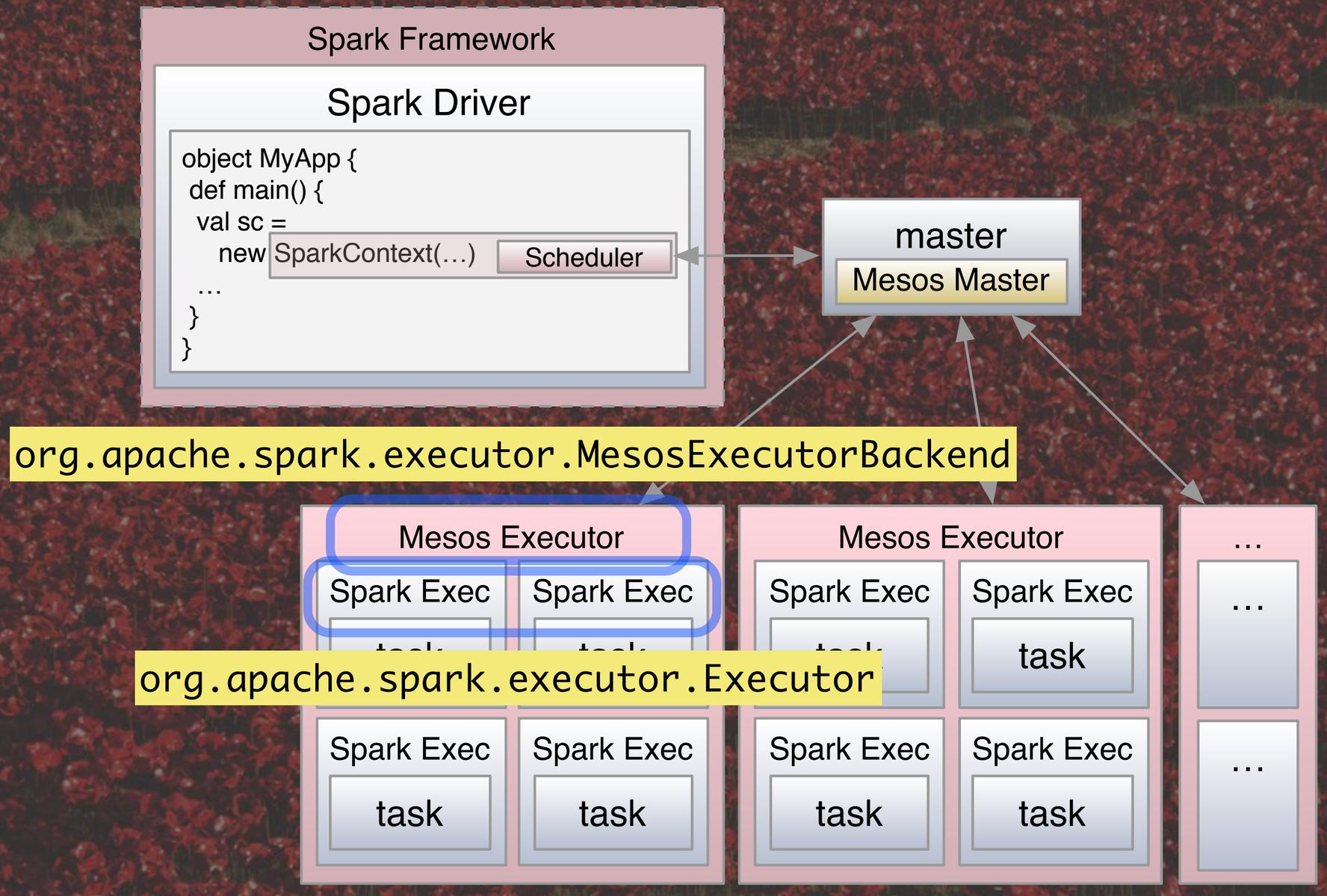
Mesos Fine Grained Mode



Tuesday, October 20, 15

There is still one Mesos executor. The actual Scala class name is now org.apache.spark.executor.MesosExecutorBackend. The nested “Spark Executor” is still the Mesos agnostic org.apache.spark.executor.Executor. The scheduler (a org.apache.spark.scheduler.cluster.mesos.MesosSchedulerBackend) is instantiated as a field in the SparkContext.

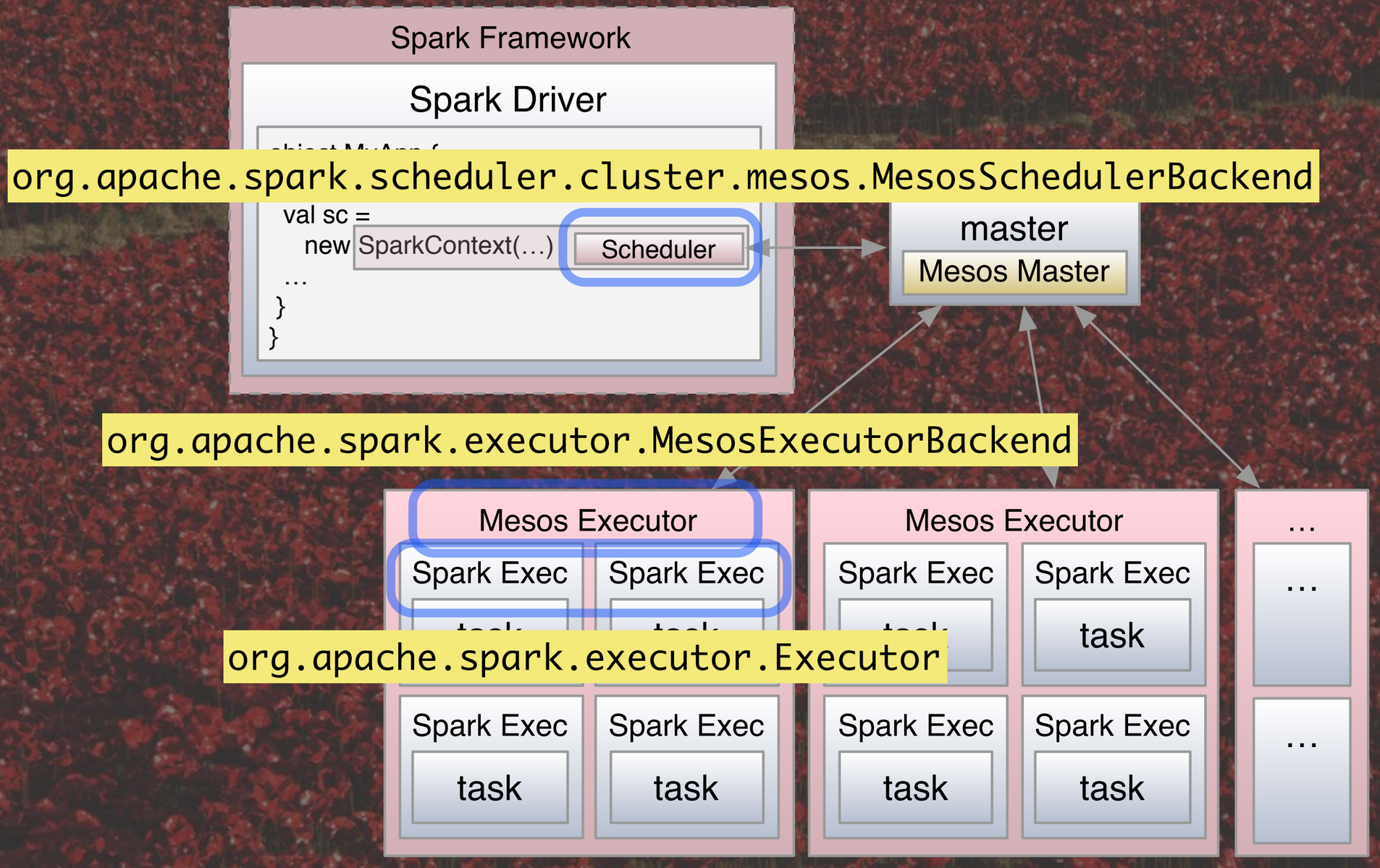
Mesos Fine Grained Mode



Tuesday, October 20, 15

There is still one Mesos executor. The actual Scala class name is now org.apache.spark.executor.MesosExecutorBackend (no “FineGrained” prefix), which is now Mesos-aware. The nested “Spark Executor” is still the Mesos-agnostic org.apache.spark.executor.Executor, but there will be one created per task now. The scheduler (a org.apache.spark.scheduler.cluster.mesos.MesosSchedulerBackend) is instantiated as a field in the SparkContext.

Mesos Fine Grained Mode

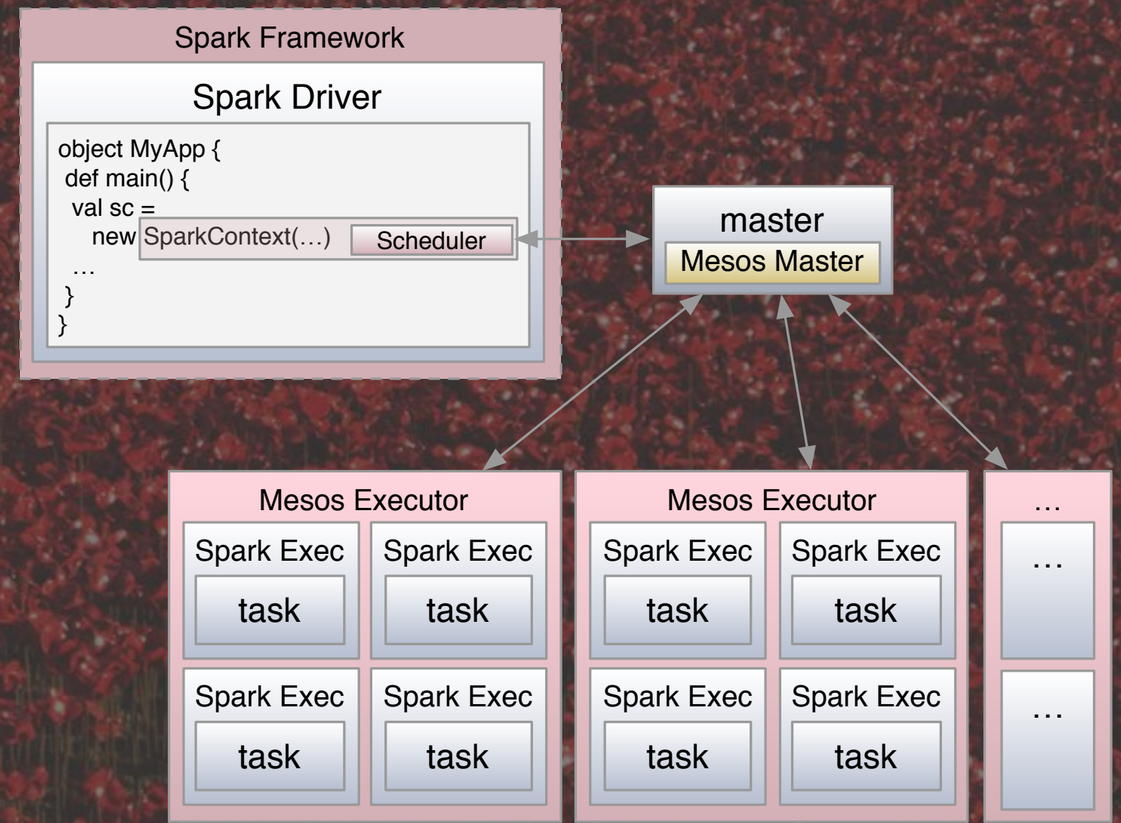


Tuesday, October 20, 15

The MesosExecutorBackend process is started by the MesosSchedulerBackend (also no “FineGrained” prefix), one per slave. As for coarse-grained mode, the MesosSchedulerBackend is a field in SparkContext.

Mesos Fine Grained Mode

- One Mesos task per Spark executor
- Spark tasks are spawned as threads.



Tuesday, October 20, 15

The isn't a separate process for the Spark executor and the underlying Spark task. Rather, the latter is run on a thread pool owned by the former.

Mesos Fine Grained Mode

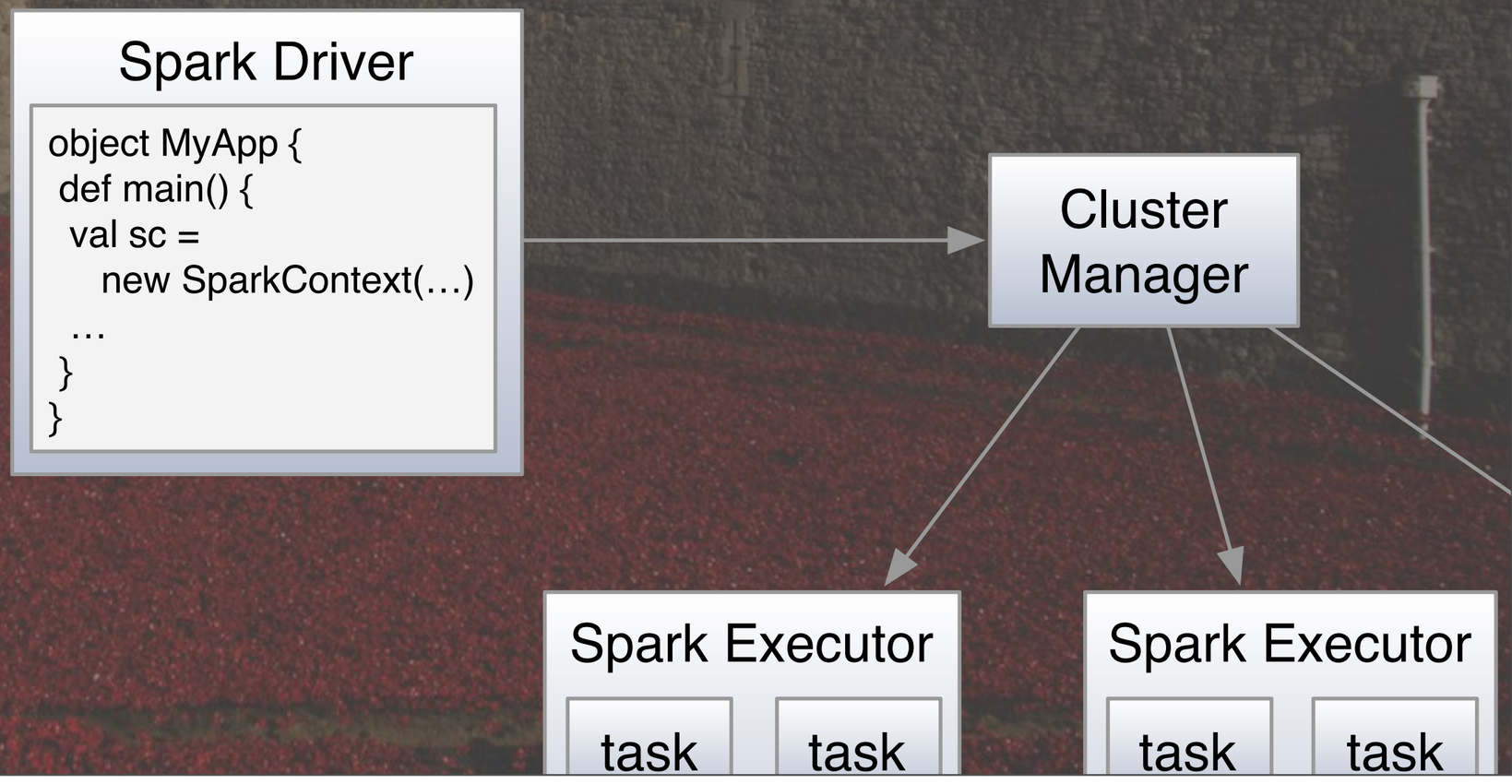
- ☐ Slower startup for tasks:
  Fine for batch and relatively static streaming.
- ☑ Better resource utilization.

Tuesday, October 20, 15

This is the Spark term for a how you run your driver program. It’s one last axis of variability...

Client vs. Cluster Mode

- Where does the *Driver* actually run?



See http://blog.cloudera.com/blog/2014/05/apache-spark-resource-management-and-yarn-app-models/ for a discussion of these concepts for YARN.

Client Mode

- Driver runs in a separate program.
- E.g., spark shell on your laptop.
- (Could also be a running process on the cluster.)
- Submitting process waits for exit.

Tuesday, October 20, 15

To be clear, if on the cluster, it's not running under the control of Mesos, YARN, etc., but perhaps you logged into a node and you're running the shell, etc.

Cluster Mode

- Driver managed by Mesos.
  - E.g., long-running streaming job.
- Submitting process exits immediately.

Near Term Future

Spark on Mesos

- Resource constraints & reservations
- Optimistic offers & oversubscription
- Framework roles & authentication
- Persistent volumes
- MOAR networking
- Log aggregation

Long Term Future?

Language?

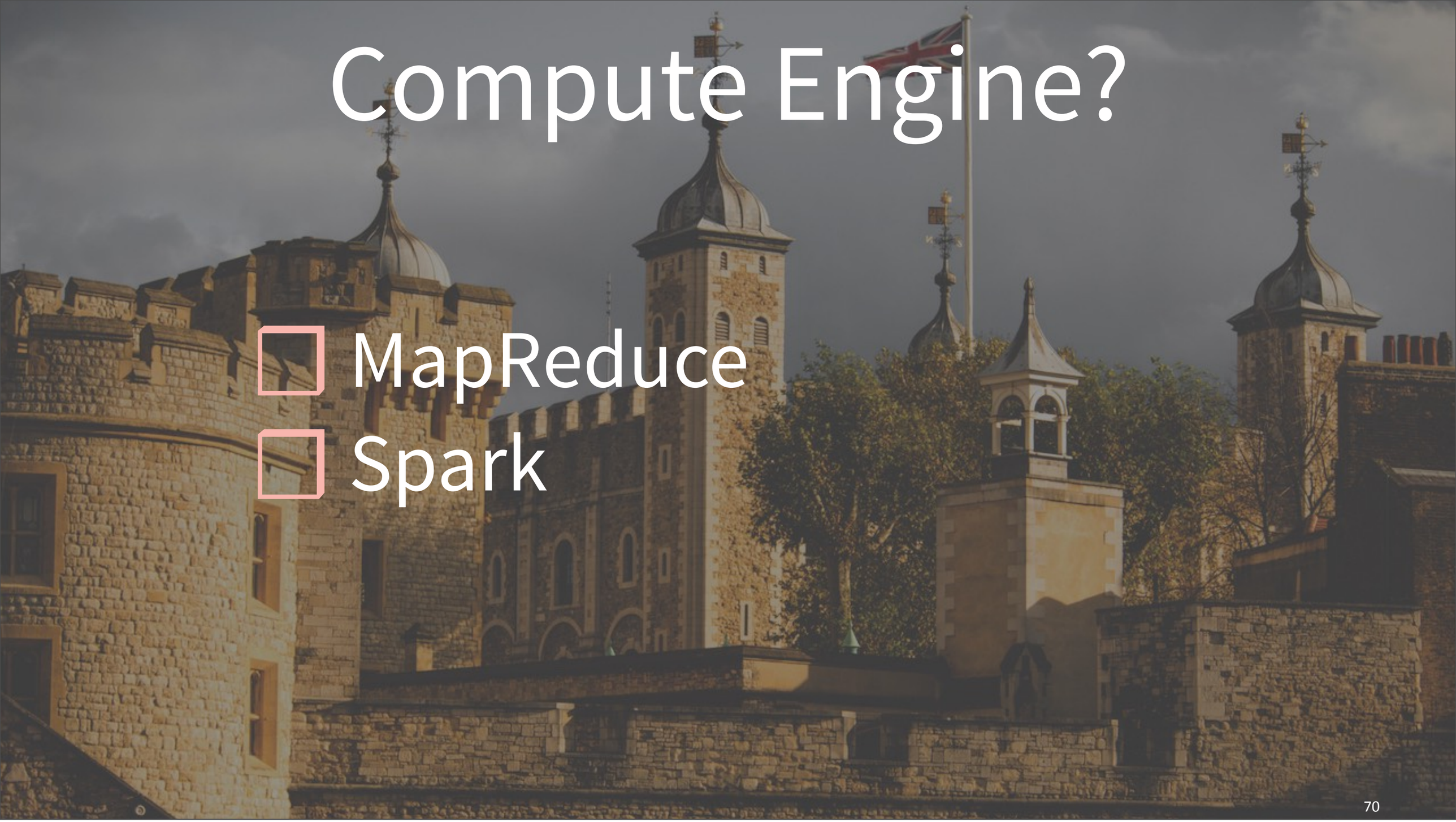
- [ ] Java
- [ ] Scala

Language?

- [ ] Java
- [x] Scala

Tuesday, October 20, 15

For Data Engineers, Scala has arguably won as the most popular language. First Scalding, then Spark, followed by other popular tools like Kafka, demonstrated how optimal a hybrid object-oriented and functional language is for building data-centric, highly scalable and resilient systems.


Compute Engine?
MapReduce
Spark

Compute Engine?

- ☐ MapReduce
- ☑ Spark

Tuesday, October 20, 15

Hat tip to Cloudera for embracing Spark when it became clear that the limitations of the first-generation MapReduce model wasn't extensible enough for modern needs.

Cluster Platform?

- [ ] YARN
- [ ] Mesos

Cluster Platform?

- [ ] YARN
- [?] Mesos

Tuesday, October 20, 15

This is cheating a bit, as I said they aren't exactly targeting the same problem and you can even run YARN on Mesos, but just as the Hadoop community replaced Spark after five or so years of widespread public use, I think five years from now Mesos could be the "baseline" for Big Data systems, including those labeled "Hadoop". Two reasons:

1. The compelling efficiency that Mesos brings and the way it lets you use one cluster, not several.
2. The relative ease with which developers are implement new services and porting existing services on top of Mesos.

Learn Product Documentation Blog

INTRODUCING MESOSPHERE INFINITY

The first integrated product enabling companies to turn ubiquitous data to insight and action.

Get Early Access

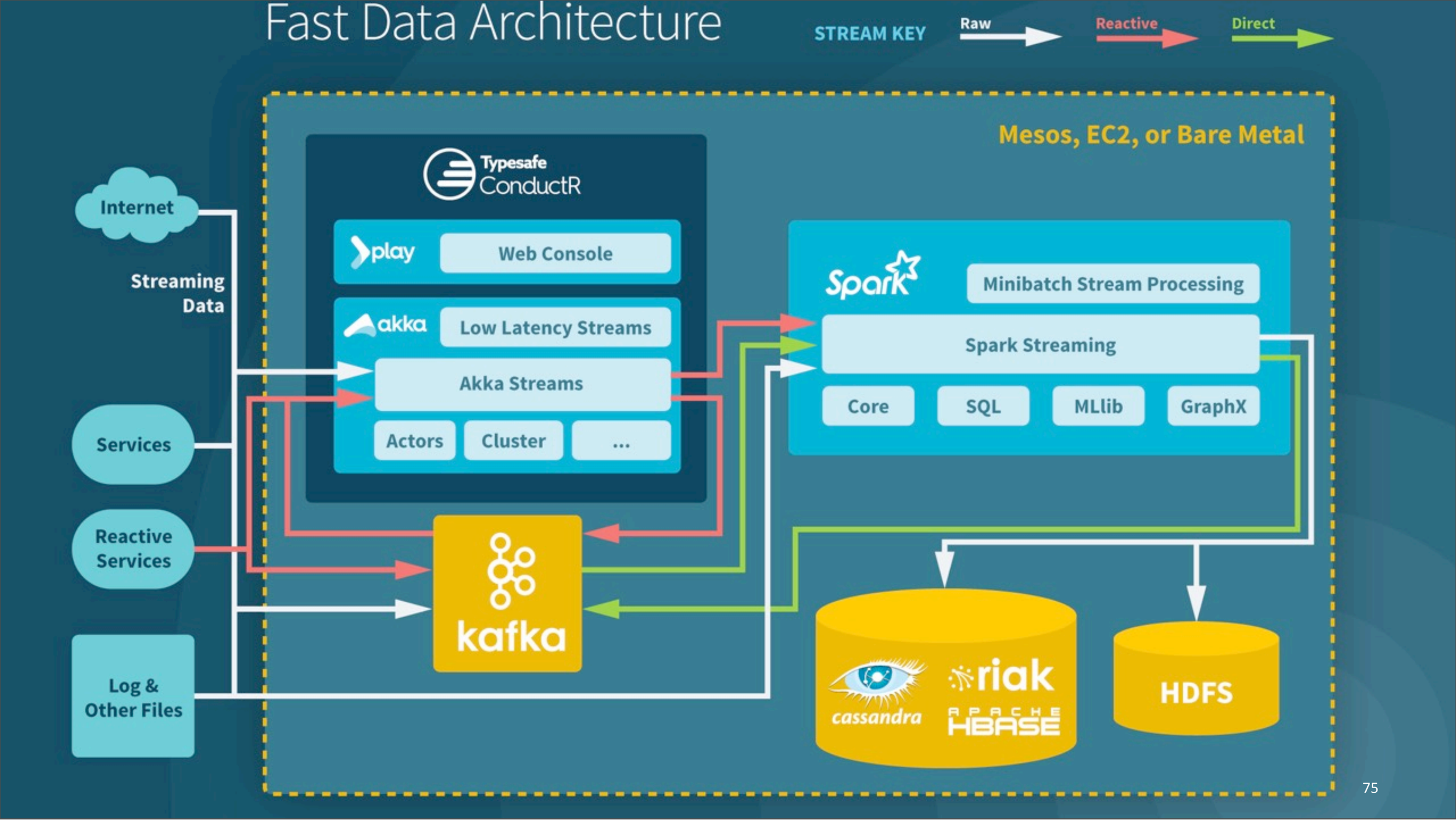
https://mesosphere.com/infinity/

OPEN ASSURED INSTANT ADAPTIVE

Tuesday, October 20, 15

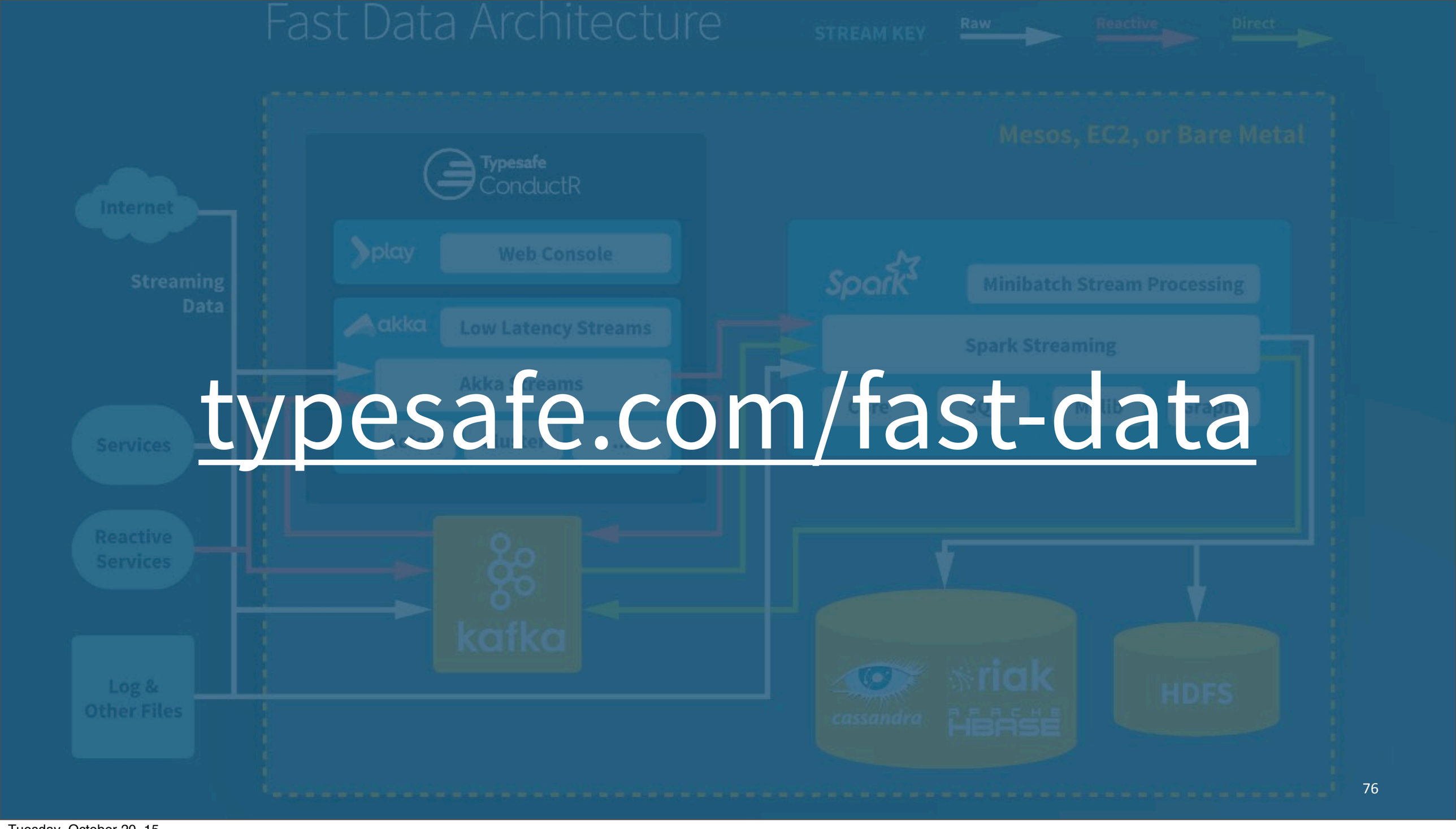
Typesafe now offers commercial support for Spark, Akka, and Scala on Mesosphere DCOS, in partnership with Mesosphere Infinity.

Fast Data Architecture



Tuesday, October 20, 15

This is my vision of the “Fast Data” architecture of the future. The link is to a whitepaper I wrote on Fast Data.



typesafe.com/fast-data

Tuesday, October 20, 15

This is my vision of the “Fast Data” architecture of the future. The link is to a whitepaper I wrote on Fast Data.

dean.wampler@typesafe.com
polyglotprogramming.com/talks
@deanwampler

Spark on Mesos