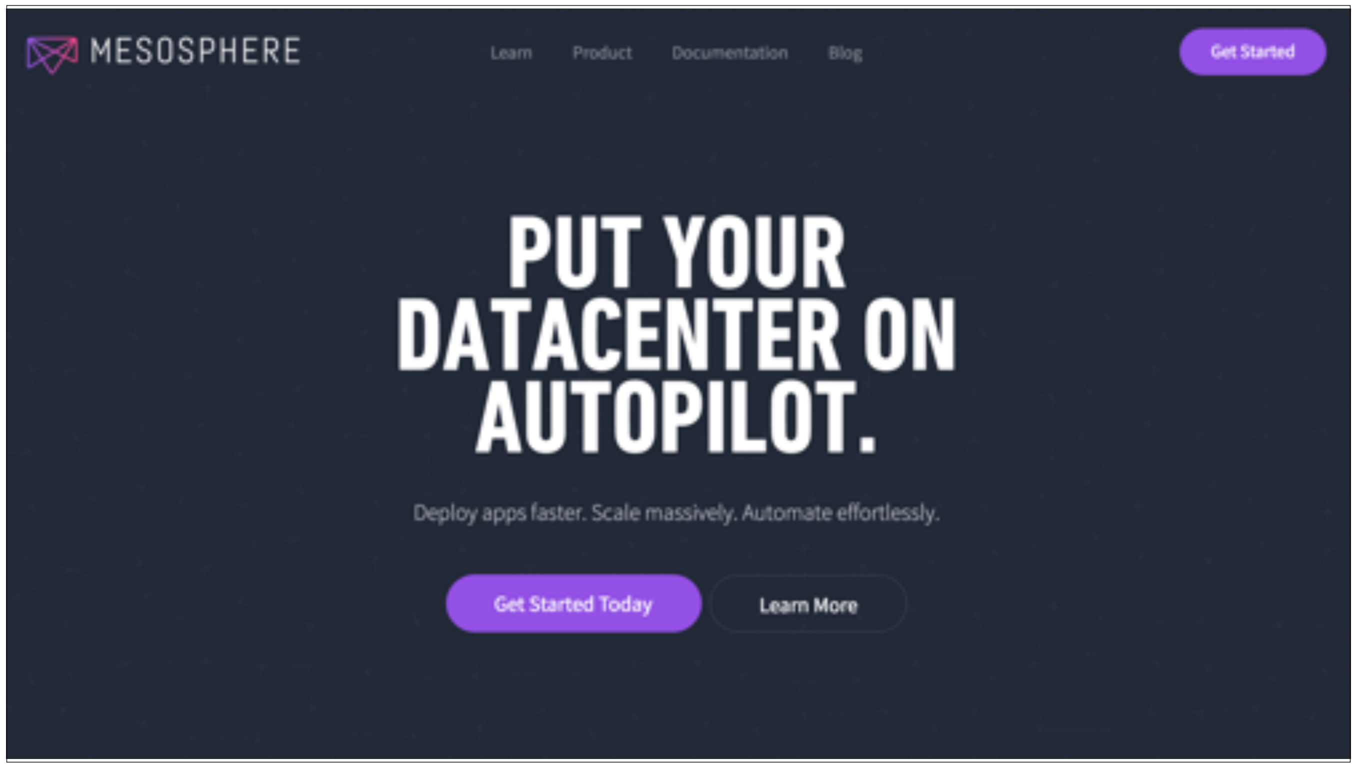
Spark on Mesos

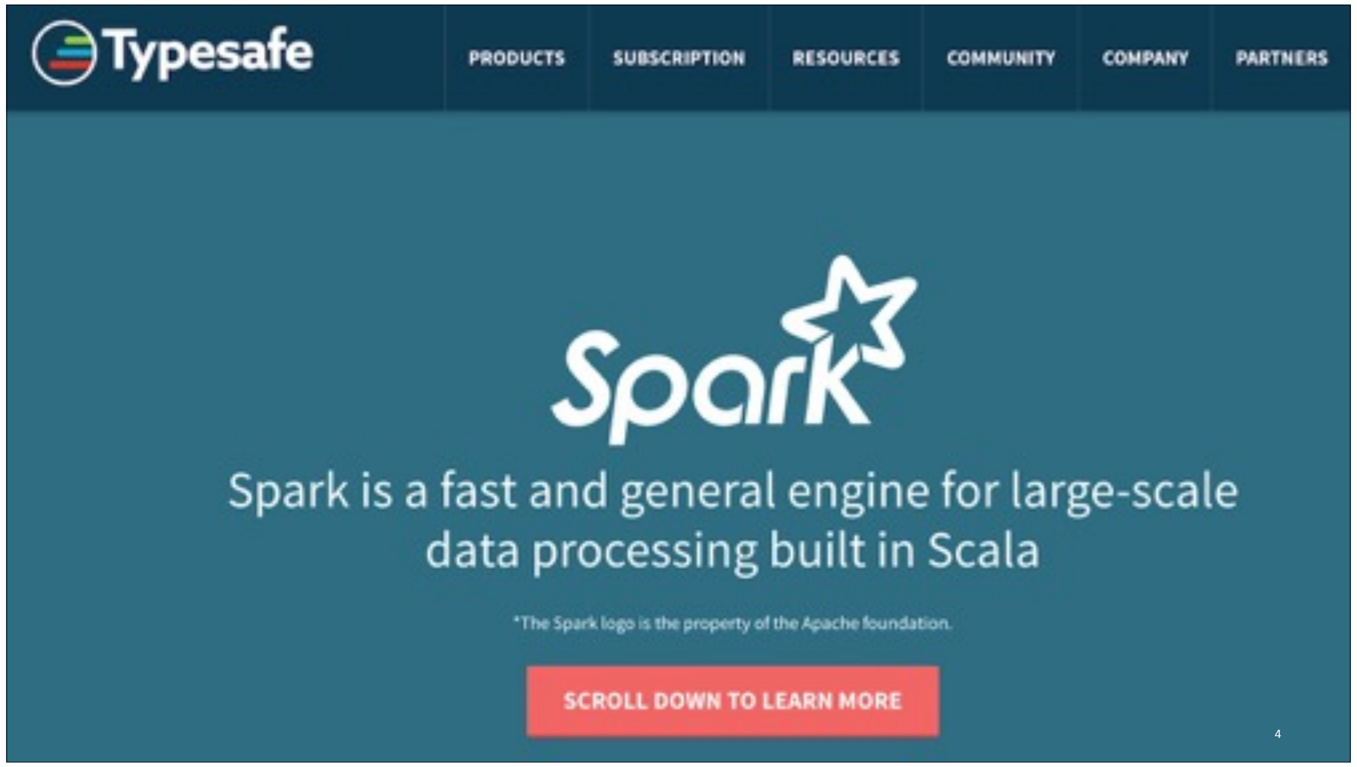
tim@mesosphere.io
@tnachen
dean.wampler@typesafe.com
@deanwampler


MESOSPHERE
Learn
Product
Documentation
Blog
Get Started
PUT YOUR DATACENTER ON AUTOPILOT.
Deploy apps faster. Scale massively. Automate effortlessly.
Get Started Today
Learn More

Mesosphere’s Data Center Operating System (DCOS) is a commercially supported Mesos ecosystem. We’ll use it in the demo of Mesos features later.

Typesafe has launched Spark support for Mesosphere DCOS

Typesafe engineers are contributing to the Mesos support. Typesafe will provide commercial support for development and production deployment. Typesafe also offers developer support for teams getting started with Spark, but with plans to deploy to other platforms, like YARN.



Typesafe

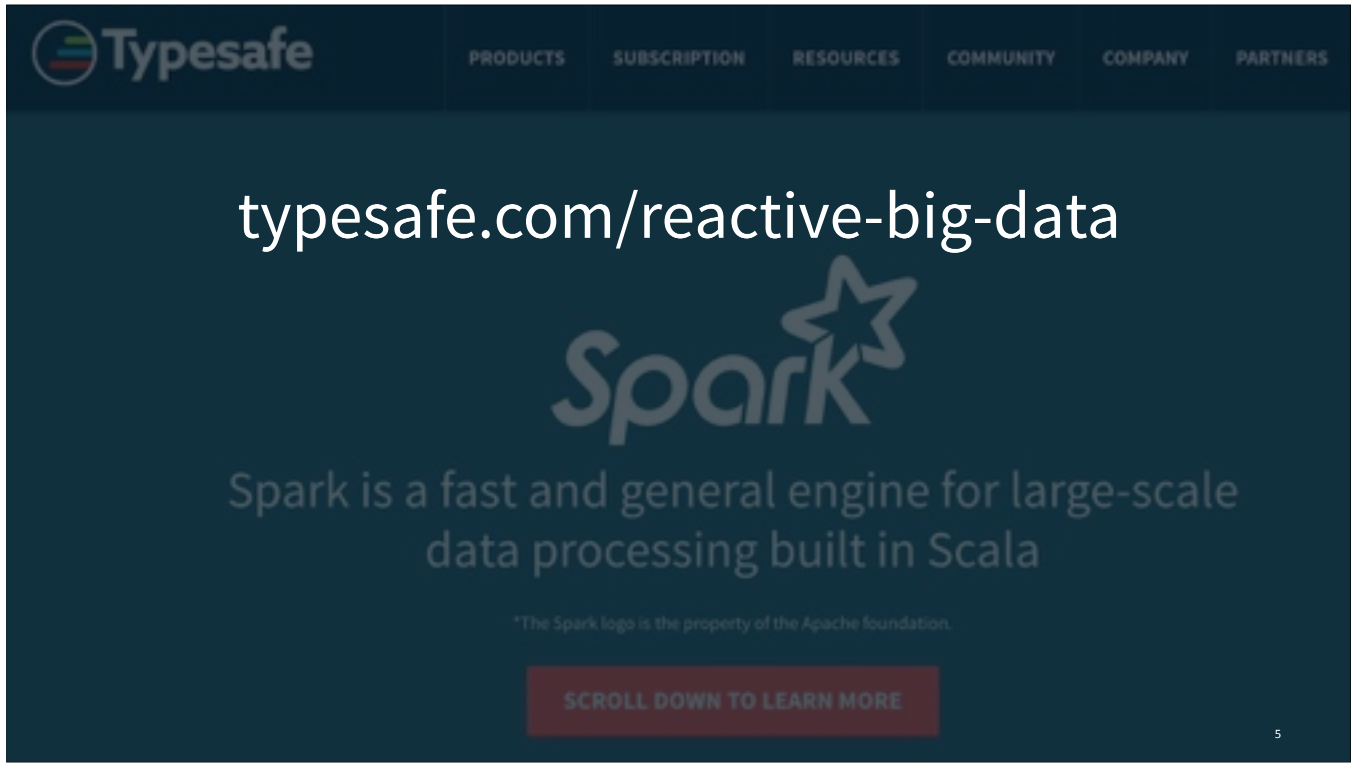
PRODUCTS SUBSCRIPTION RESOURCES COMMUNITY COMPANY PARTNERS

Spark is a fast and general engine for large-scale data processing built in Scala

*The Spark logo is the property of the Apache foundation.

SCROLL DOWN TO LEARN MORE

This page provides information about what Typesafe is doing with Spark, including our support offerings, the results of a recent survey of Spark usage, blog posts and webinars about the world of Spark.



typesafe.com/reactive-big-data

Typesafe

PRODUCTS SUBSCRIPTION RESOURCES COMMUNITY COMPANY PARTNERS

Spark is a fast and general engine for large-scale data processing built in Scala

*The Spark logo is the property of the Apache foundation.

SCROLL DOWN TO LEARN MORE

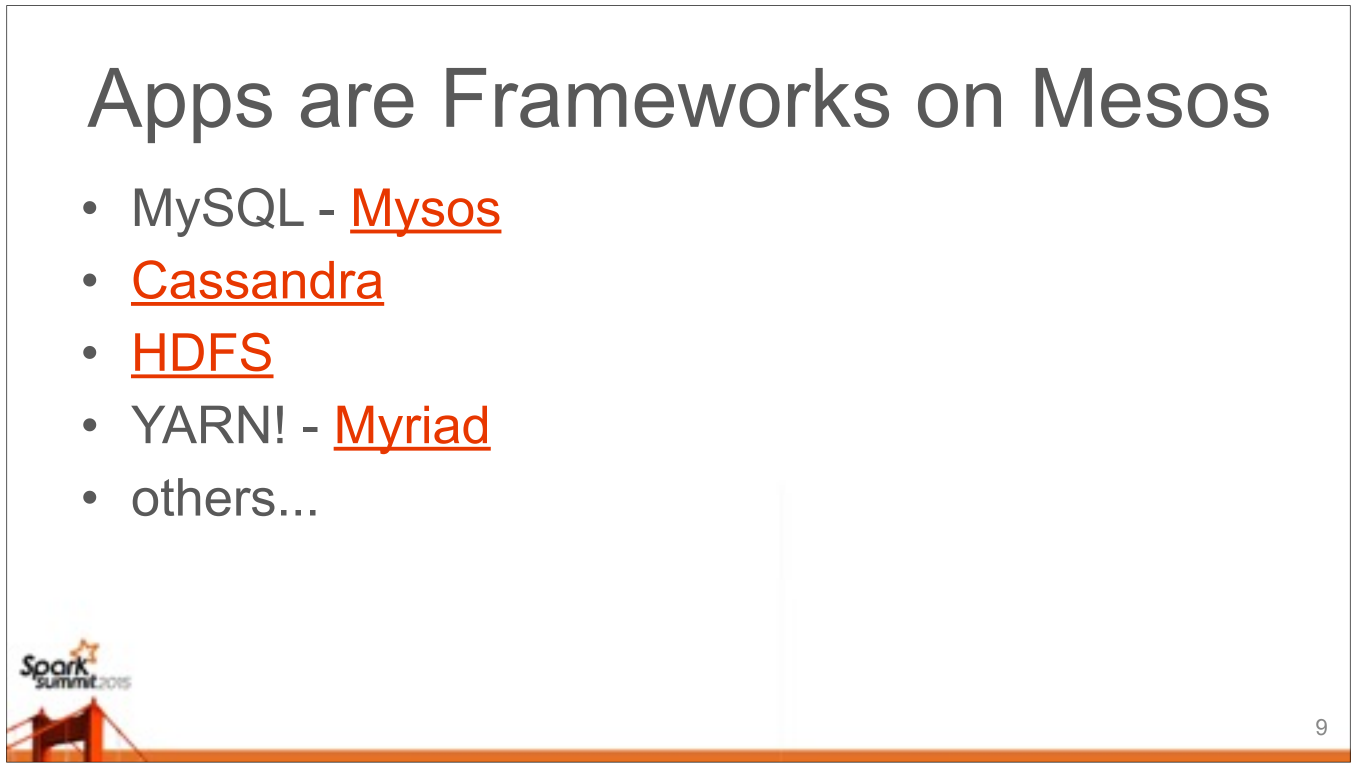
This page provides more information, as well as results of a recent survey of Spark usage, blog posts and webinars about the world of Spark.

Mostly, we're about helping you navigate treacherous waters…
http://petapixel.com/2015/06/15/raccoon-photographed-riding-on-an-alligators-back/

Mesos

mesos.apache.org

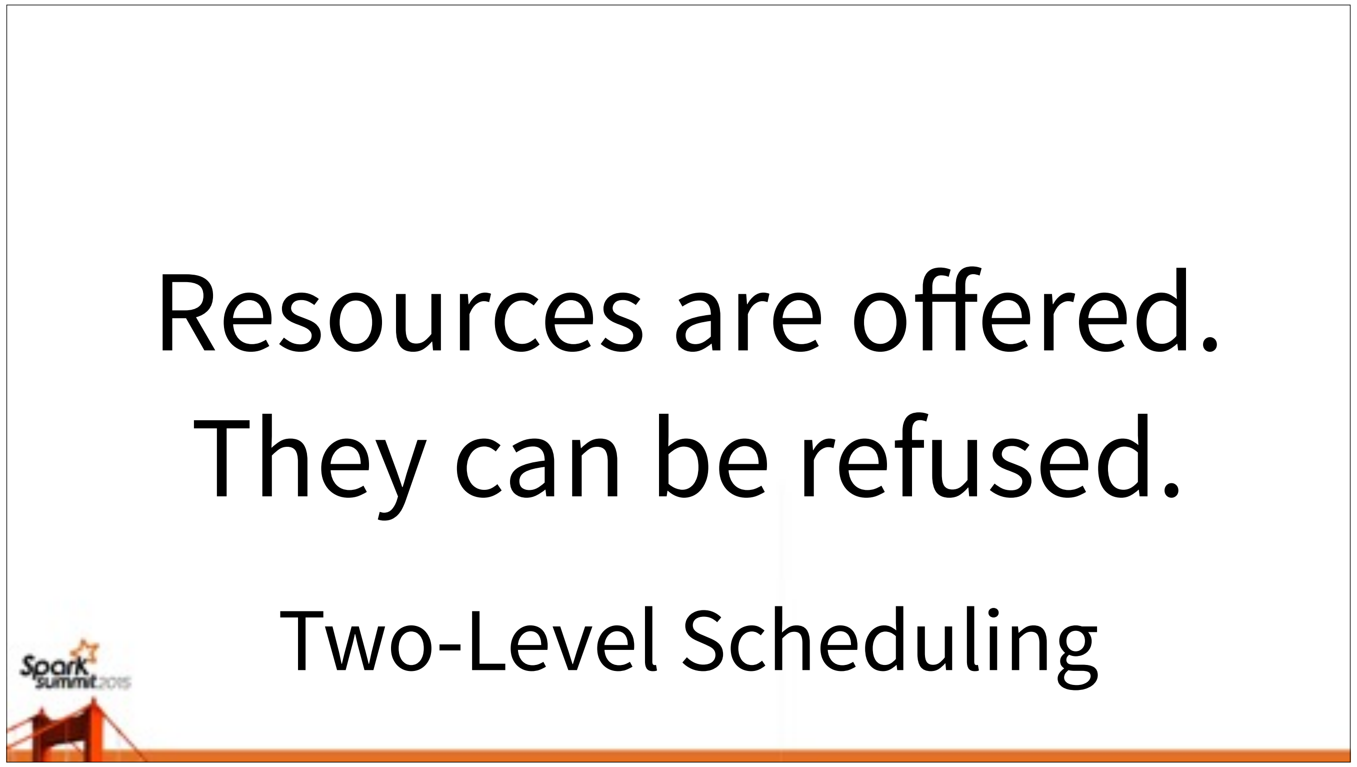
Mesos’ flexibility has made it possible for many frameworks to be supported on top of it. For example, the third generation of Apple’s Siri now runs on Mesos.



Apps are Frameworks on Mesos

- MySQL - Mysos
- Cassandra
- HDFS
- YARN! - Myriad
- others...

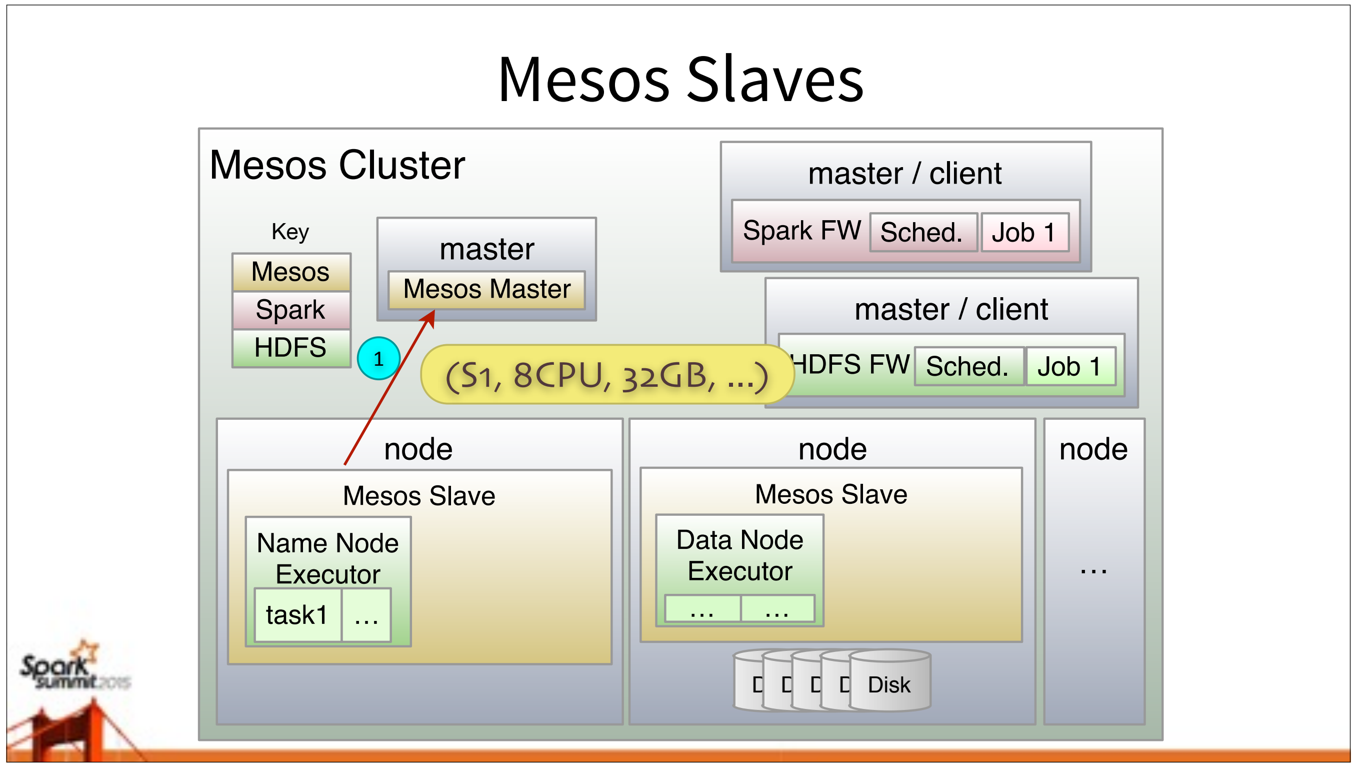
Mesos' flexibility has made it possible for many frameworks to be supported on top of it. For more examples, see http://mesos.apache.org/documentation/latest/mesos-frameworks/ Myriad is very interesting as a bridge technology, allowing (once it's mature) legacy YARN-based apps to enjoy the flexible benefits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



Resources are offered. They can be refused.

Two-Level Scheduling

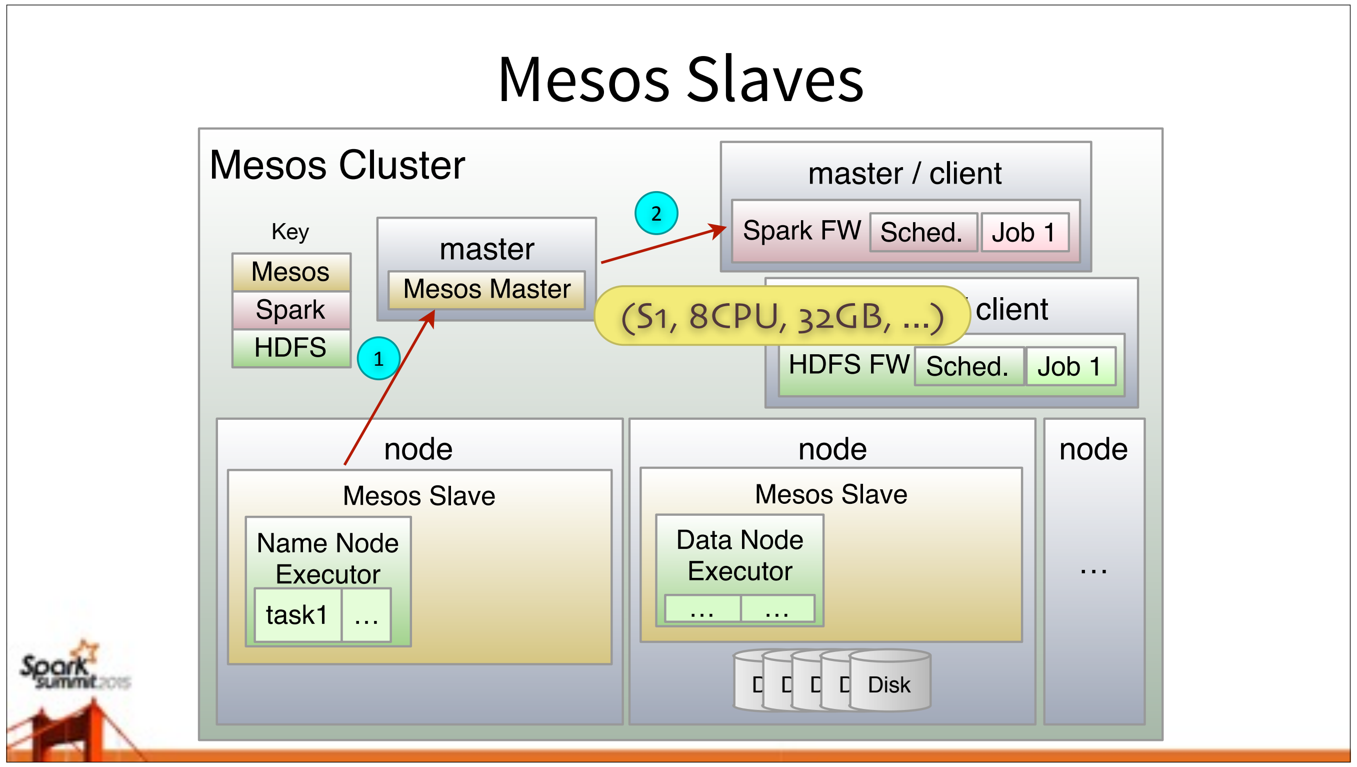
A key strategy in Mesos is to offer resources to frameworks, which can chose to accept or reject them. Why reject them? The offer may not be sufficient for the need, but it’s also a technique for delegating to frameworks the logic for imposing policies of interest, such as enforcing data locality, server affinity, etc.
Resources are dynamic and include CPU cores, memory, disk, & ports.
Scheduling and resource negotiation fine grained and per-framework.



Mesos Slaves

Here we show HDFS 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.)

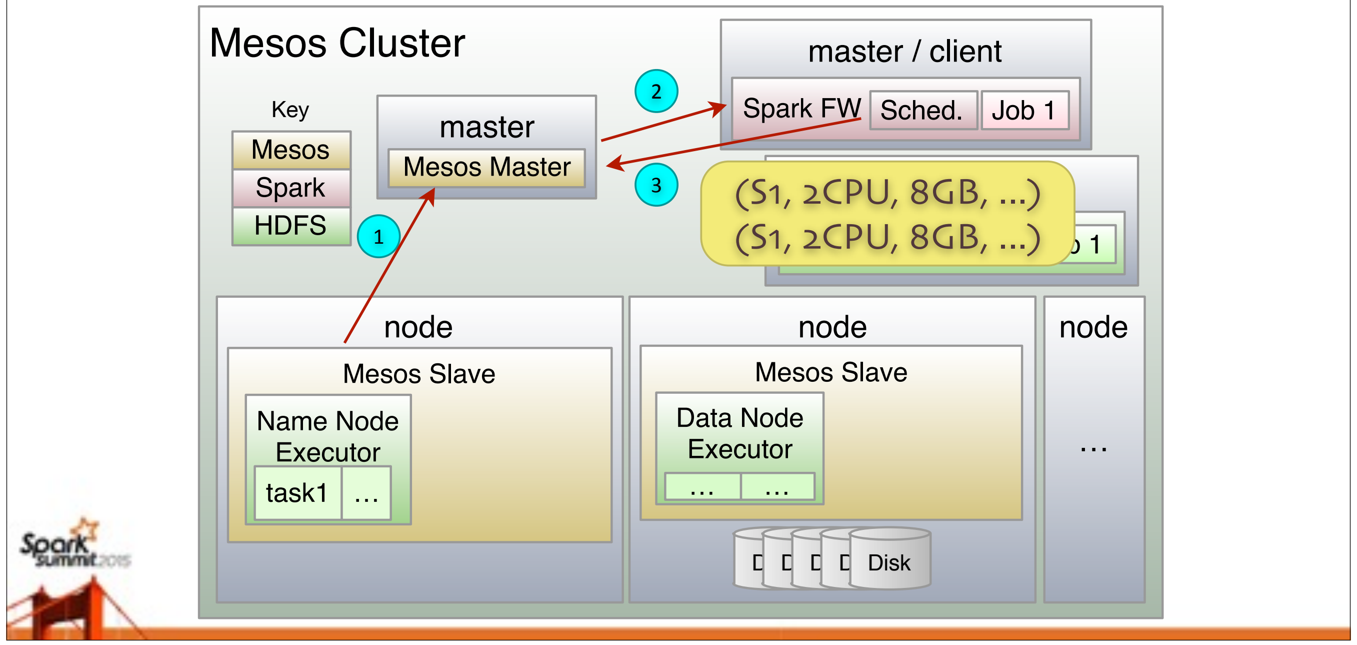
Adapted from http://mesos.apache.org/documentation/latest/mesos-architecture/



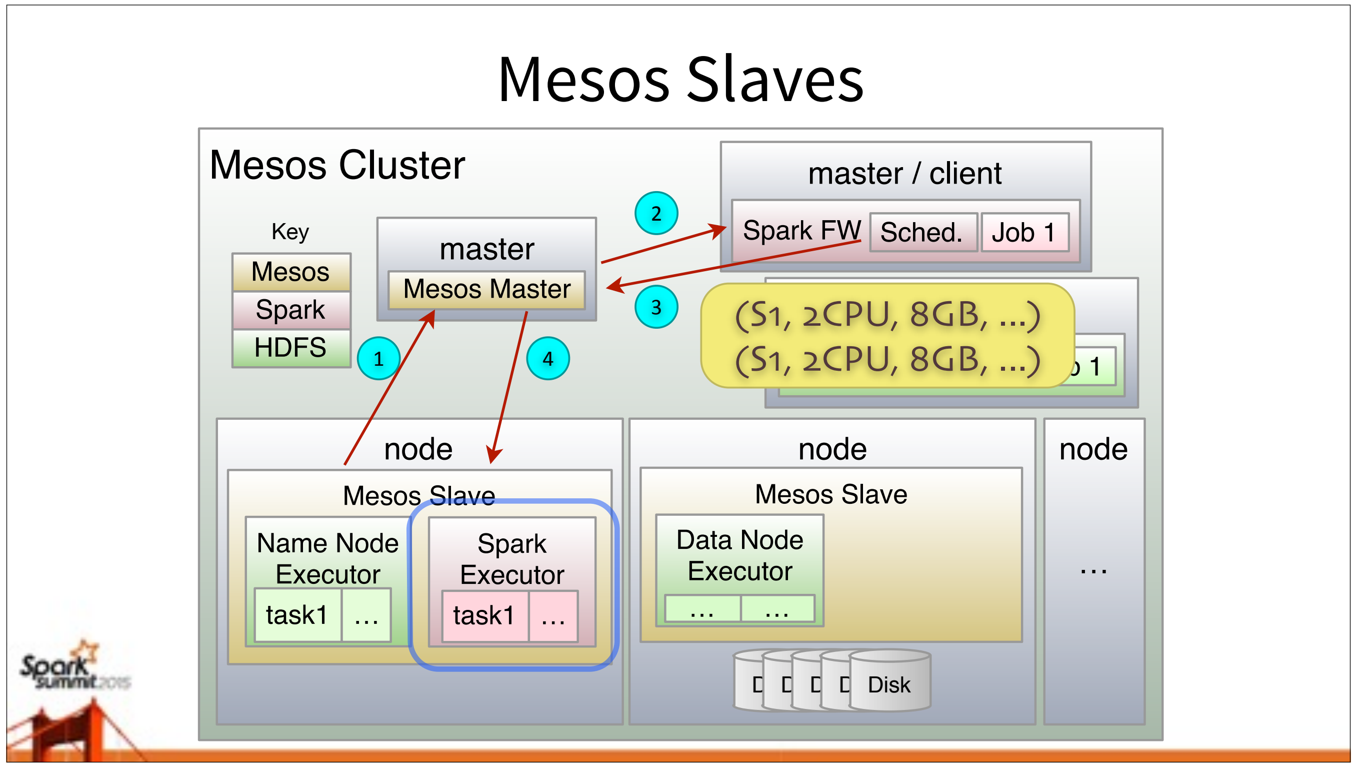
Mesos Slaves

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

Mesos Slaves

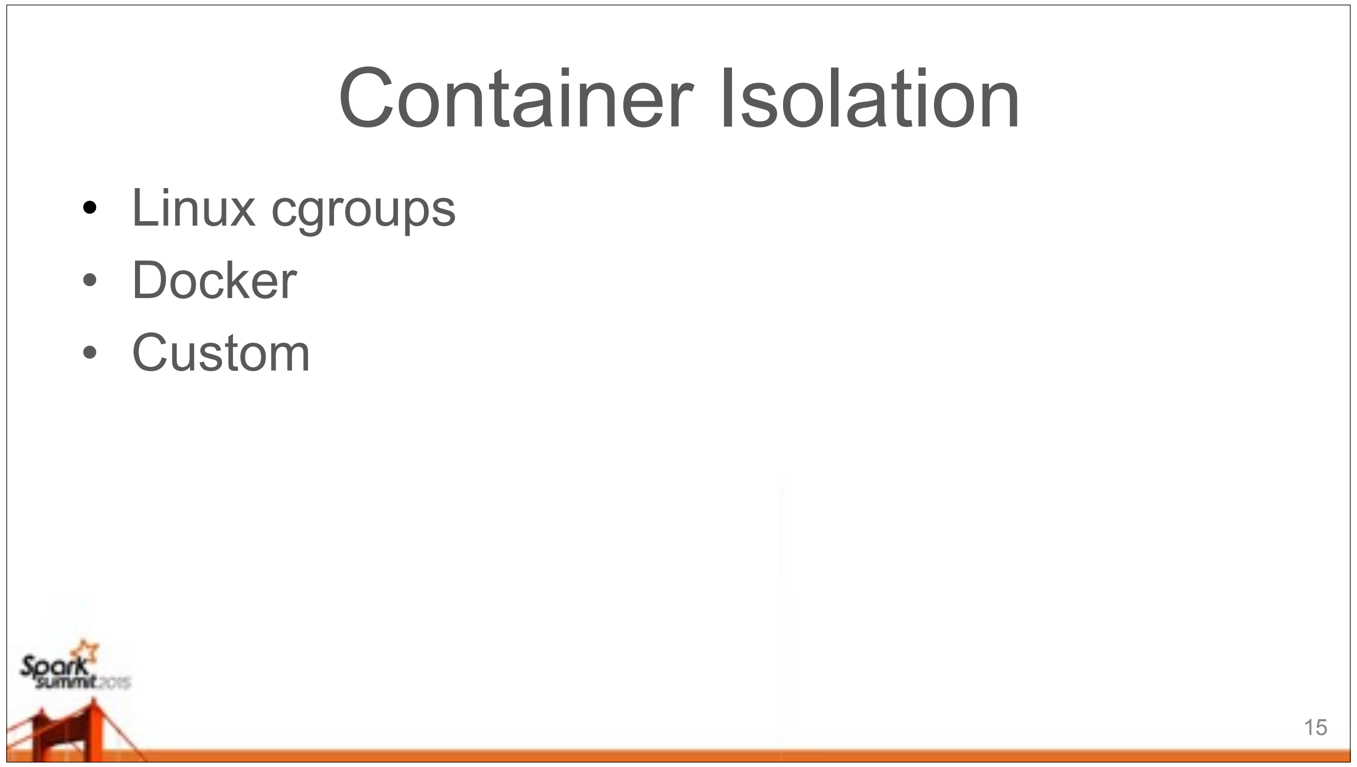


3. The Spark Framework Scheduler replies to the Master to run two tasks on the node, each with 2 CPU cores and 8GB of memory. The Master can then offer the rest of the resources to other Frameworks.



Mesos Slaves

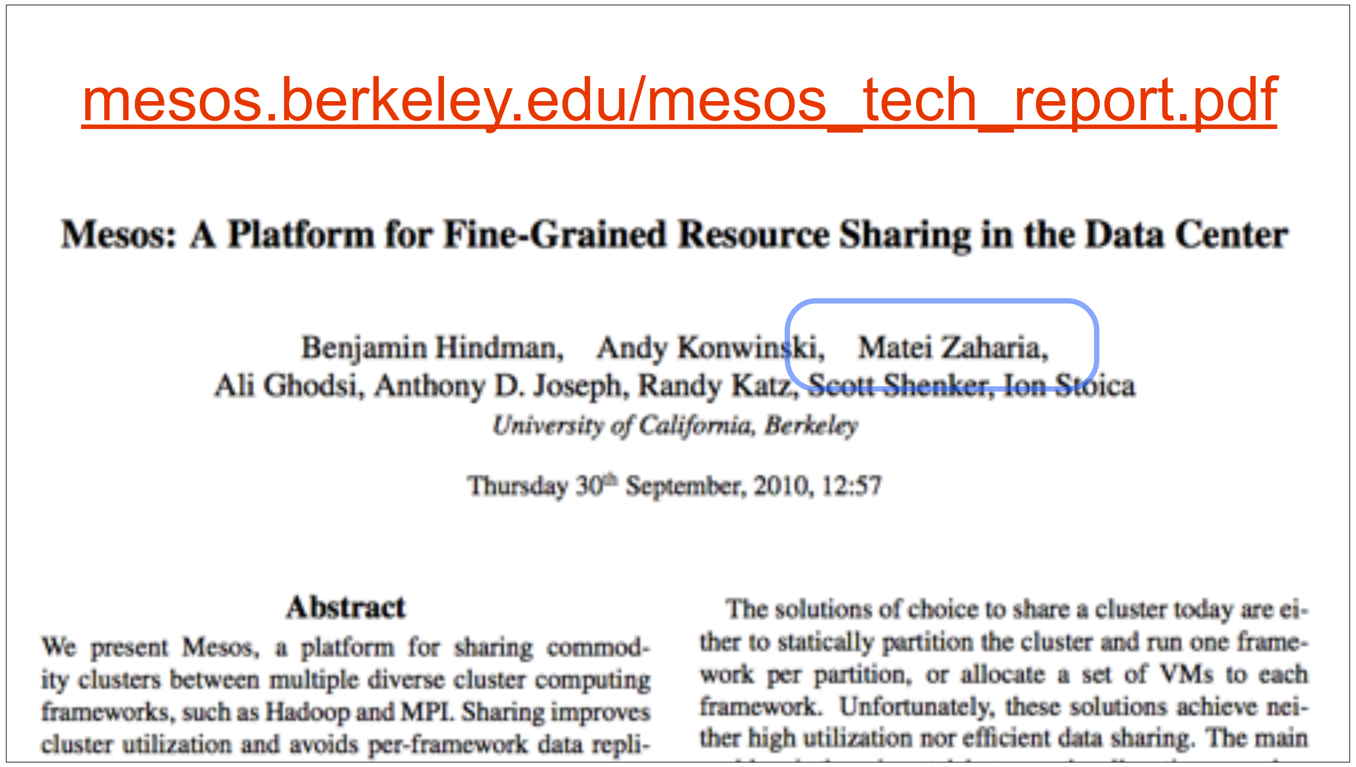
4. The master spawns the executor (if not already running - we'll dive into this bubble!!) and the subordinate tasks.



Container Isolation

- Linux cgroups
- Docker
- Custom

Last point, Mesos also gives you flexible options for using containers to provide various levels of isolation and packaging, including abstractions for defining your own container model.



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 repli-

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

For more details, it’s worth reading the very clear research paper by Benjamin Hindman, the creator of Mesos, Matei Zaharia, the creator of Spark, and others.

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

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 repli-

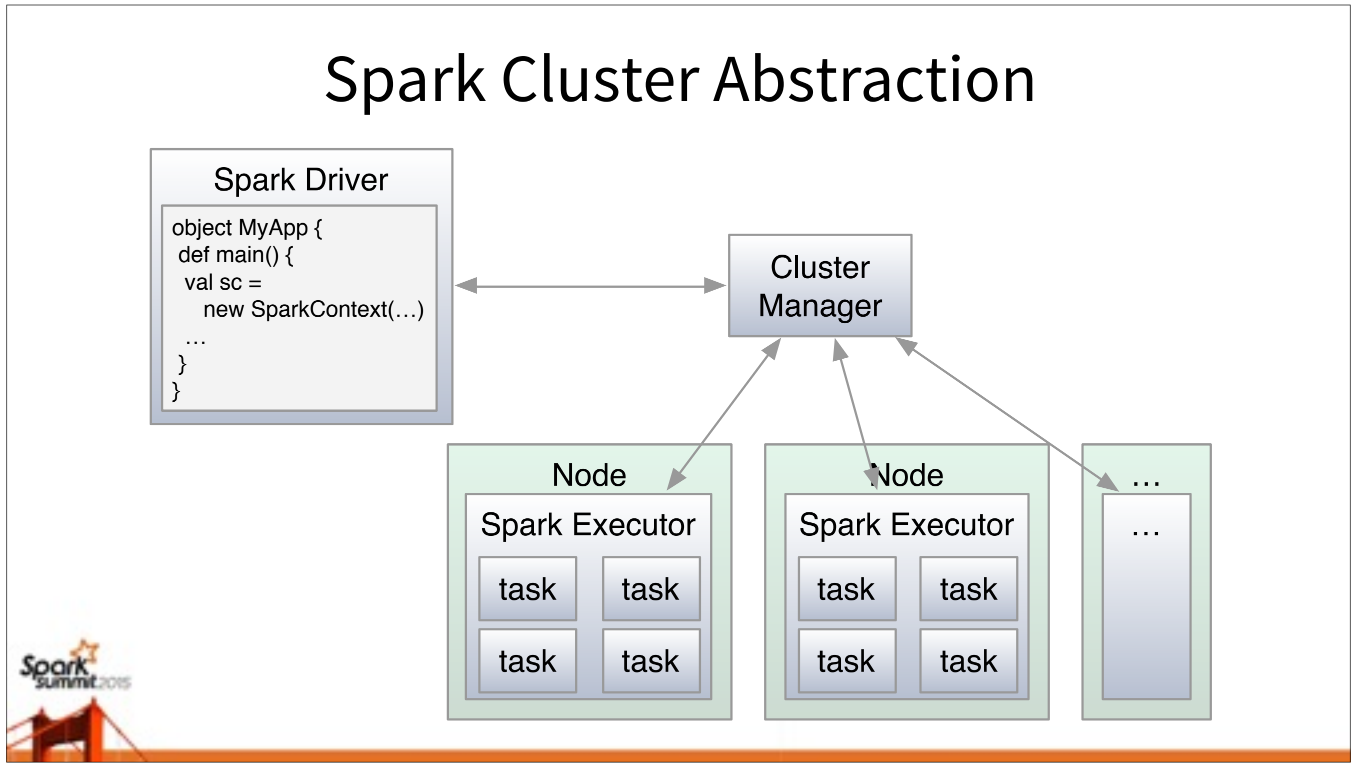
The solutions of choice to share a cluster today are either to statically partition the cluster and run one framework per partition, or allocate a set of VMs to each framework. Unfortunately, these solutions achieve neither high utilization nor efficient data sharing. The main

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

This quote is particular interesting…

Spark on Mesos

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

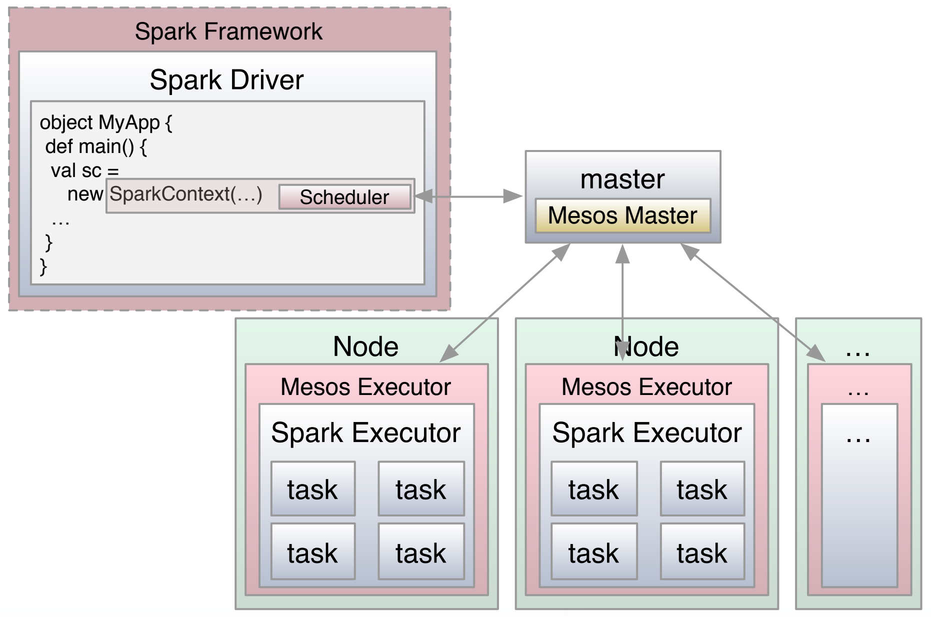


Spark Cluster Abstraction

```
object MyApp {
 def main() {
  val sc =
     new SparkContext(…)
  …
 }
}
```

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.

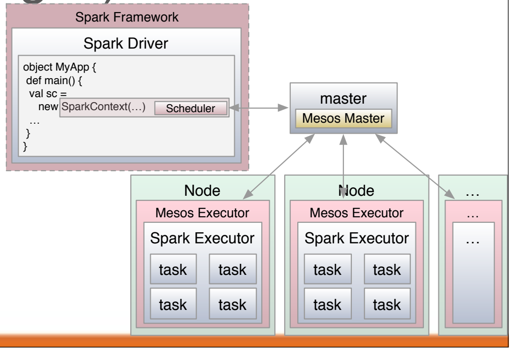
Mesos Coarse Grained Mode



Unfortunately, because Spark and Mesos "grew up together", each uses the same terms for concepts that have diverged. The Mesos and Spark "executors" are different. In Spark, 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… One CoarseMesosSchedulerBackend instance is created by the SparkContext as a field in the instance.

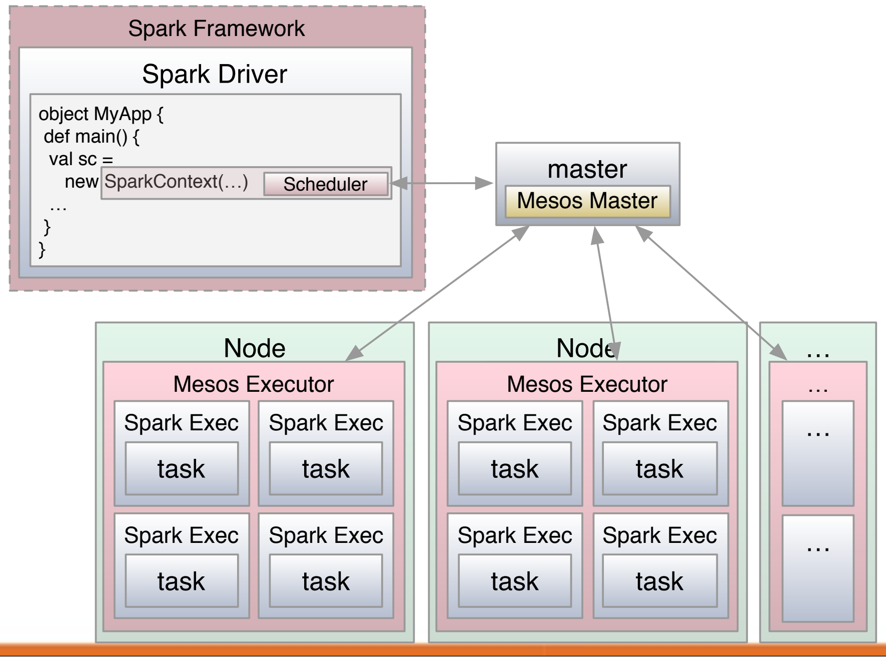
Mesos Coarse Grained Mode

- Fast startup for tasks:
  - Better for interactive sessions.
- But resources locked up in larger Mesos task.
  - (Dynamic allocation is coming…)



Tradeoffs of coarse-grained mode.

Mesos Fine Grained Mode

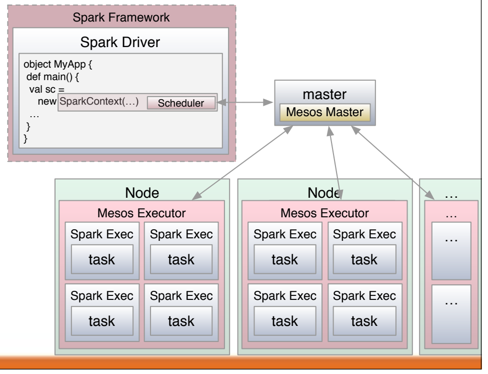


```
object MyApp {
 def main() {
  val sc =
    new SparkContext(…)
  …
 }
}
```

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

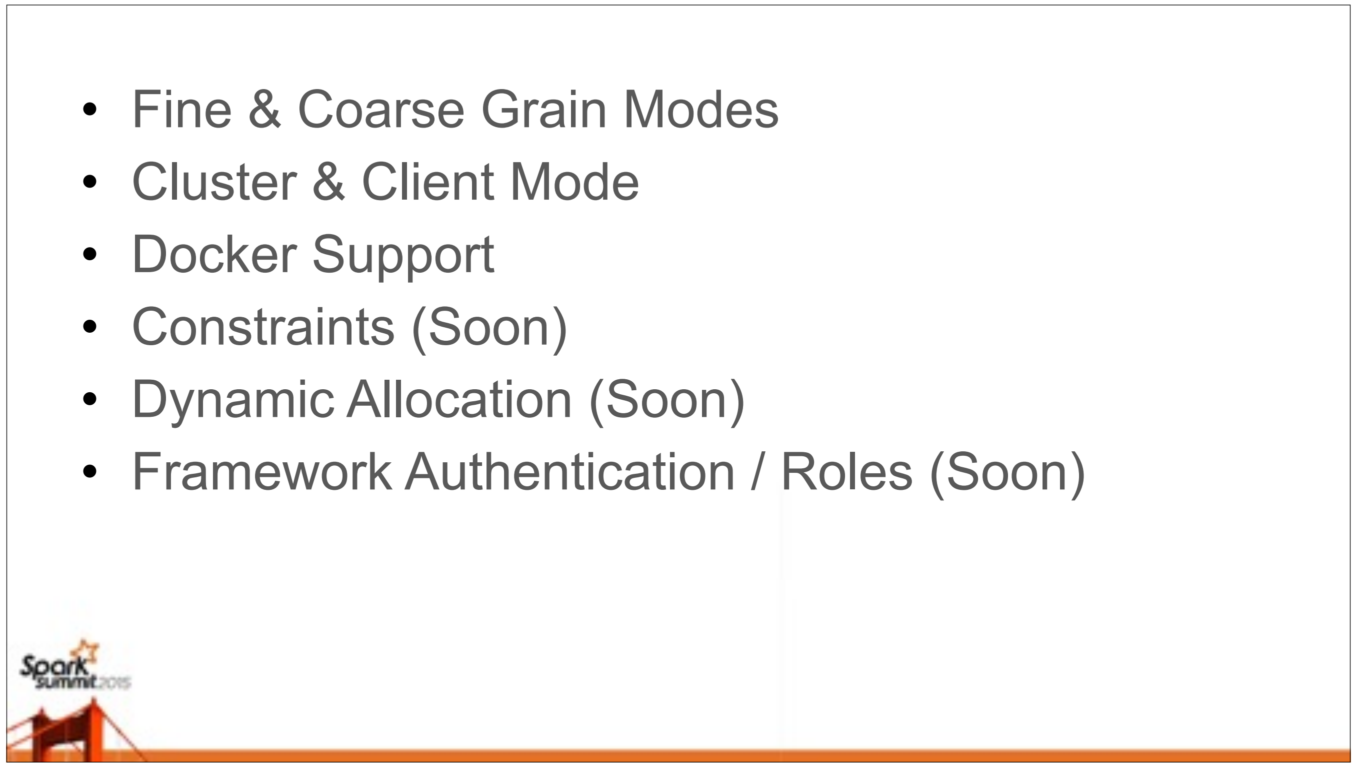
- Better resource utilization.
- Slower startup for tasks:
  - Fine for batch and relatively static streaming.



```
object MyApp {
 def main() {
  val sc =
    new SparkContext(…)
  …
 }
}
```

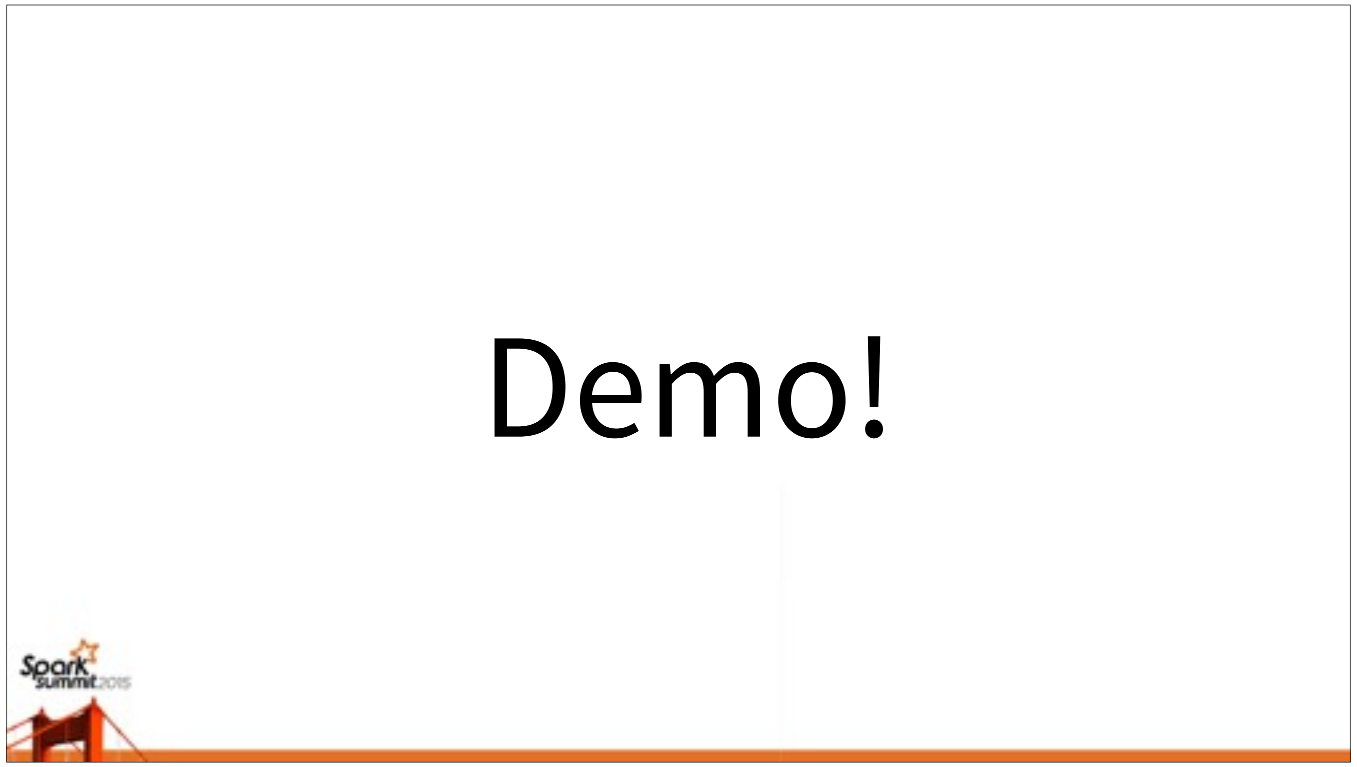
Tradeoffs

Recap



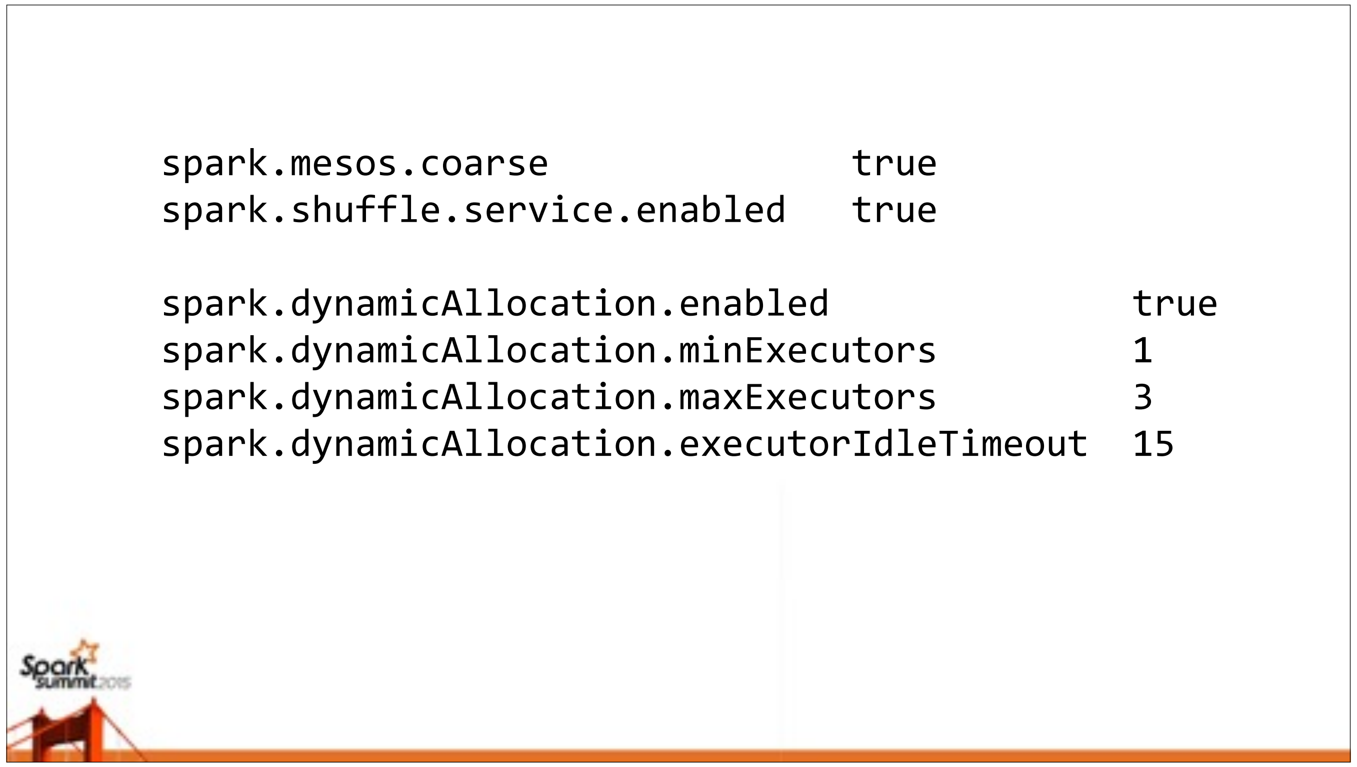
- Fine & Coarse Grain Modes
- Cluster & Client Mode
- Docker Support
- Constraints (Soon)
- Dynamic Allocation (Soon)
- Framework Authentication / Roles (Soon)

“Soon” means not yet merged into Spark master.



Demo!

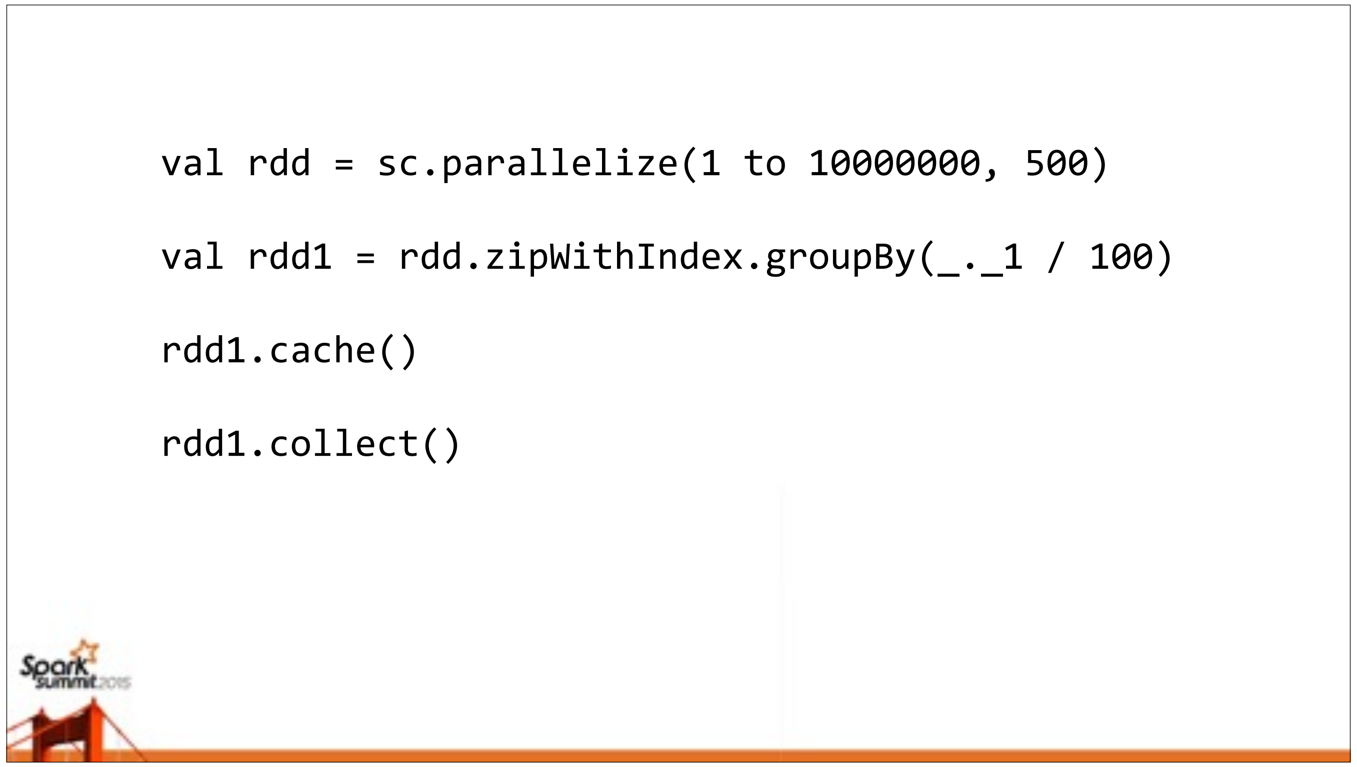
Dean will demo supervision, which restarts a job automatically if it crashes or another problem happens. In this case, the docker image will disappear.



```
spark.mesos.coarse              true
spark.shuffle.service.enabled   true

spark.dynamicAllocation.enabled                 true
spark.dynamicAllocation.minExecutors            1
spark.dynamicAllocation.maxExecutors            3
spark.dynamicAllocation.executorIdleTimeout   15
```

Not demoed, but another feature that will be merged into Spark soon for Mesos is dynamic allocation, where idle resources are reclaimed after a user-specified timeout (15 secs. here - which is probably too short for actual production). This is what you would put in spark-defaults.conf to turn on dynamic allocation, set the timeout, etc.



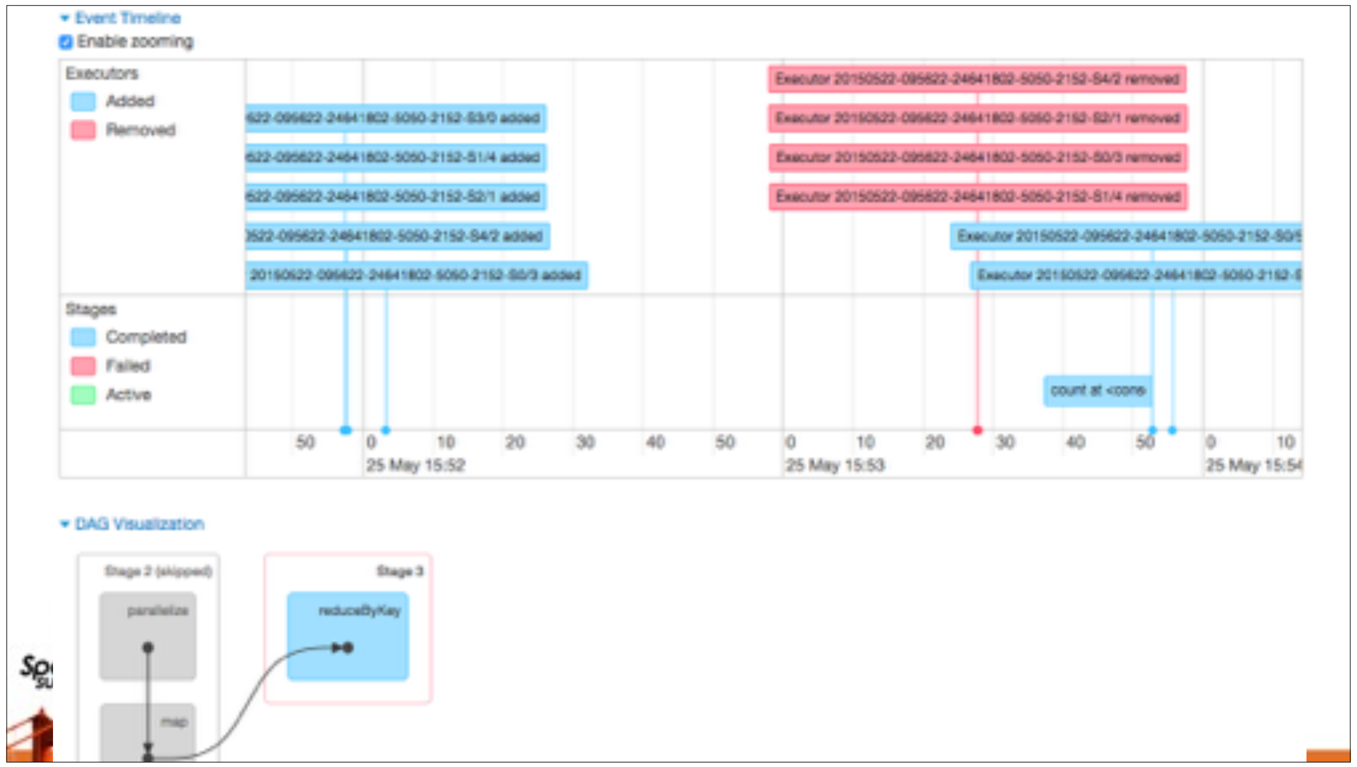
```
val rdd = sc.parallelize(1 to 10000000, 500)

val rdd1 = rdd.zipWithIndex.groupBy(_._1 / 100)

rdd1.cache()

rdd1.collect()
```

The feature can be demonstrated with a simple script in spark-shell. Run this, this do nothing for 15 seconds…


Event Timeline
Enable zooming
Executors
Added
Removed
Executor 20150522-095622-24641802-5050-2152-S4/2 removed
Executor 20150522-095622-24641802-5050-2152-S2/1 removed
Executor 20150522-095622-24641802-5050-2152-S0/3 removed
Executor 20150522-095622-24641802-5050-2152-S1/4 removed
522-095622-24641802-5050-2152-S3/0 added
522-095622-24641802-5050-2152-S1/4 added
522-095622-24641802-5050-2152-S2/1 added
522-095622-24641802-5050-2152-S4/2 added
20150522-095622-24641802-5050-2152-S0/3 added
Executor 20150522-095622-24641802-5050-2152-S0/5
Executor 20150522-095622-24641802-5050-2152-S
Stages
Completed
Failed
Active
count at <cons
50
0
10
20
30
40
50
0
10
20
30
40
50
0
10
25 May 15:52
25 May 15:53
25 May 15:54
DAG Visualization
Stage 2 (skipped)
parallelize
map
Stage 3
reduceByKey

… And spark kills the idle executors. If you do more work, it starts new executors. We're also running the separate shuffle service here. This means that Spark can reuse the shuffle files output from Stage 2, without having to repeat that part of the pipeline (grey color), before doing Stage 3 (blue).

What's Next for Mesos?

- Oversubscription
- Persistence Volumes
- Networking
- Master Reservations
- Optimistic Offers
- Isolations
- More….

Thanks!

dean.wampler@typesafe.com
@deanwampler
tim@mesosphere.io
@tnachen