# Understanding Spark Application Performance with Spark's Web UI


# Lesson Objectives

- After completing this lesson, you should be able to:
  - Understand how concepts like cores, executors, tasks and stages apply to actual Spark applications
  - Be able to use the Spark Web UI to understand how your Spark application works
  - Be able to find the running time of each job, stage, and task

# Spark Web UI

## Monitor running applications

```
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder.
  master("local[4]"). //start with 4 cores
  appName("My Spark Application").
  getOrCreate()
```

This lesson uses Spark 2.0.0

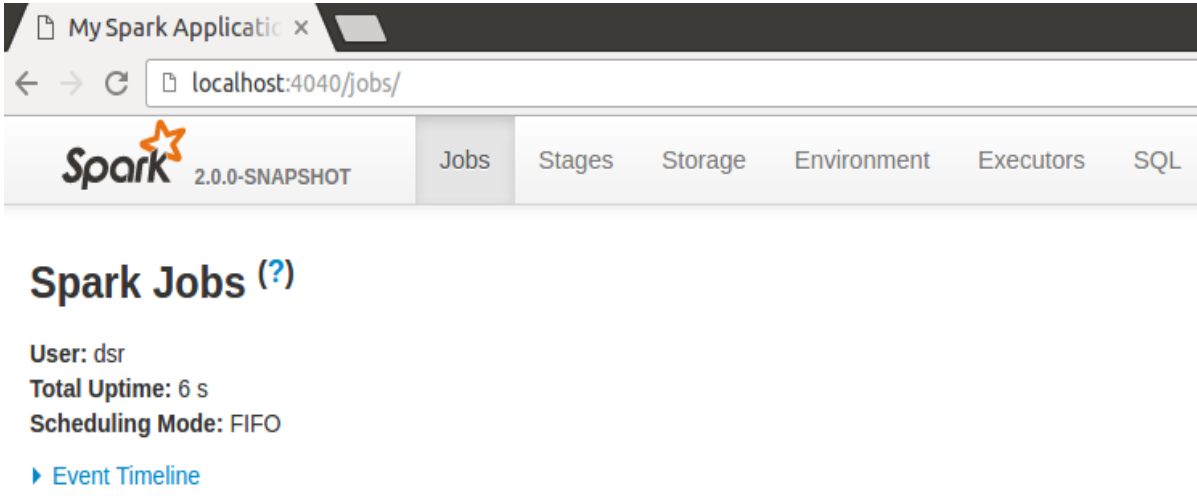
```
16/07/06 13:32:27 INFO Utils: Successfully started service 'SparkUI' on port 4040.
16/07/06 13:32:27 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.1.100:4040
```

```
sparkSession.sparkContext.uiWebUrl
```

```
Some(http://192.168.1.100:4040)
```

# Spark Web UI

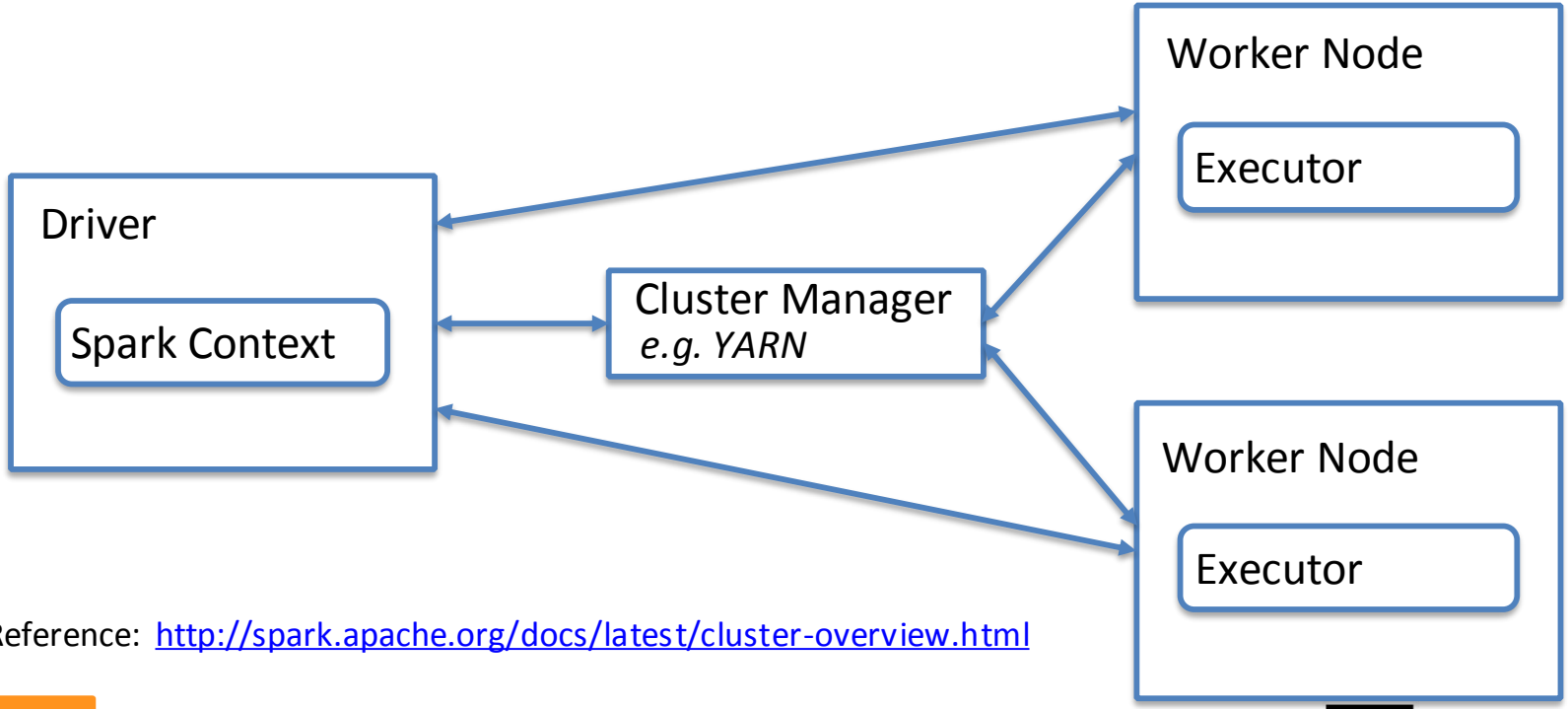
- Environment
- Jobs
- Stages
- Tasks
- Cache (Storage)
- SQL
- Streaming




Lightbend

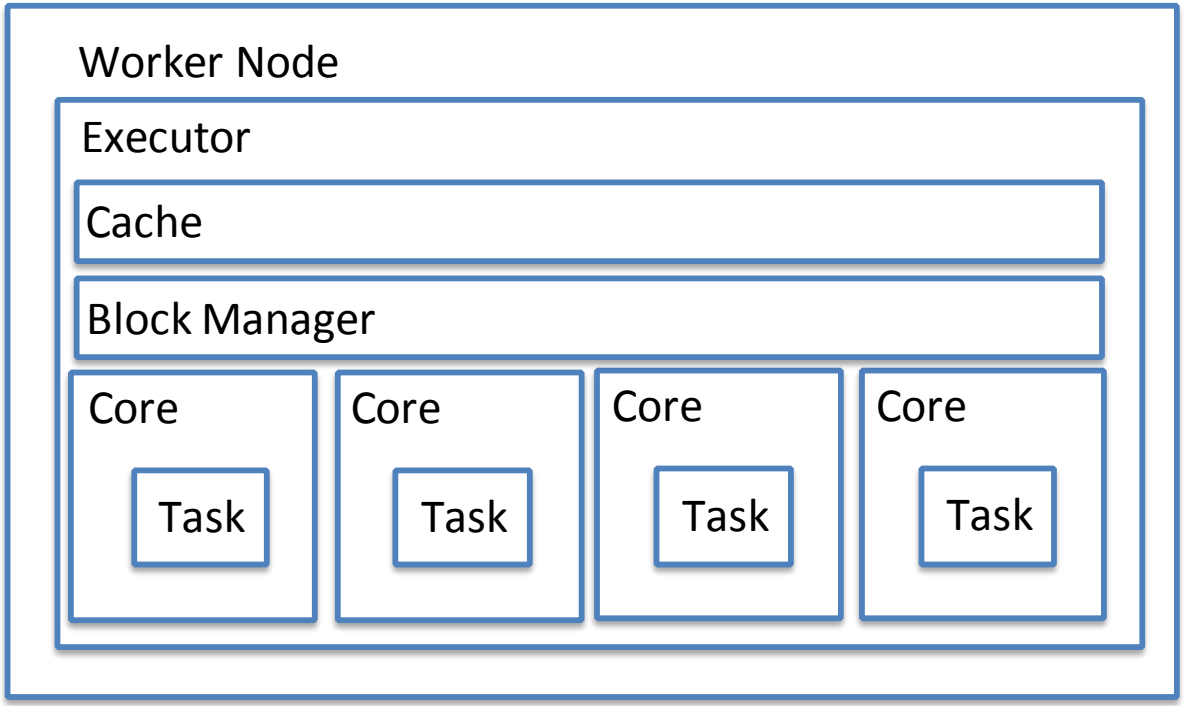
DATA SCIENCE RETREAT®

# Spark Architecture




Reference: http://spark.apache.org/docs/latest/cluster-overview.html

# Inside a Worker Node




Lightbend

DATA SCIENCE RETREAT®

# Executor and Cores in Web UI

```
import org.apache.spark.sql.SparkSession

val sparkSession = SparkSession.builder.master("local[4]")
```

My Spark Applicati ×

localhost:4040/executors/

Spark 2.0.0-SNAPSHOT | Jobs | Stages | Storage | Environment | Executors | SQL | My Spark Applic

## Executors

### Summary

| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Active(1) | 0 | 0.0 B / 755.3 MB | 0.0 B | 4 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| Dead(0) | 0 | 0.0 B / 0.0 B | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| Total(1) | 0 | 0.0 B / 755.3 MB | 0.0 B | 4 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |

### Executors

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | 127.0.1.1:36855 | Active | 0 | 0.0 B / 755.3 MB | 0.0 B | 4 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | Thread Dump |

Lightbend

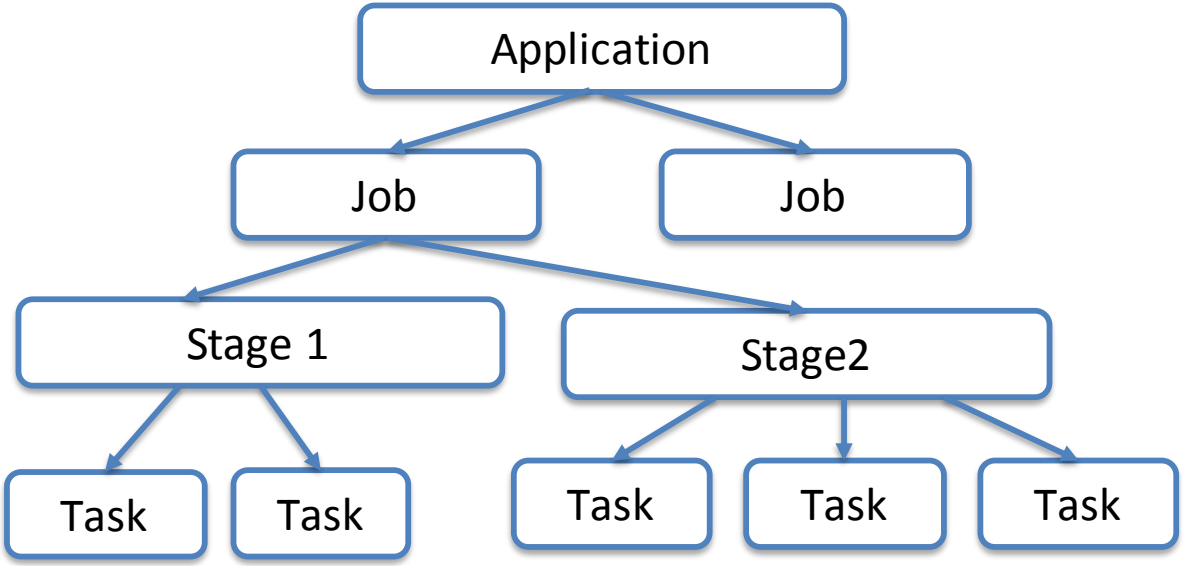
DATA SCIENCE RETREAT®

# Spark Application Life Cycle

- Create a Spark context and obtain executors
- Spark application calls an action (such as collect() or count())
- Spark creates an execution schedule that consists of tasks
  - From DAG to stages to tasks
- The driver program asks the cluster manager to run tasks on executors
- Executors run the tasks
- The driver program collects the results
- The application stops the Spark context

Reference: https://spark.apache.org/docs/latest/job-scheduling.html

# Jobs, Stages and Tasks




Spark Web UI shows breakdown of a running application

Lightbend

DATA SCIENCE RETREAT®

# Spark Program Example

```
val sparkSession = . . .
val trans = sparkSession.sparkContext
  .parallelize(List(1,2,3,4),
    numSlices = 4).map(_ + 1)
val output = trans.collect()
```

- Create a Spark context
- Use 4 partitions
- Define transformations (e.g. map, filter)
- Materialize result by applying an action (e.g. count, collect)

Lightbend

DATA SCIENCE RETREAT®

# Jobs

User script:

```
val trans = sparkSession.sparkContext
 .parallelize(List(1,2,3,4),
   numSlices = 4).map(_ + 1)
val output = trans.collect()
```

Implementation of collect:

```
def collect(): Array[T] = {
  val results = sc.runJob(/*...*/)
  //...
}
```


Lightbend

DATA SCIENCE RETREAT®

# Stages

```
val trans = ...
    parallelize(...).
    map(...)
trans.toDebugString
```

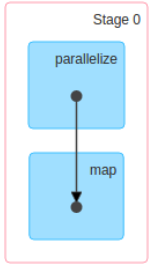
```
(4) MapPartitionsRDD[1] at
        map at <console>:13 []
 |  ParallelCollectionRDD[0] at
        parallelize at
        <console>:13 []
```

Spark 2.0.0-SNAPSHOT | Jobs | Stages | Storage | Environment | Executors | SQL

## Details for Job 0

**Status:** SUCCEEDED
**Completed Stages:** 1

- Event Timeline
- DAG Visualization




### Completed Stages (1)

| Stage Id | Description | Submitted | Duration | Tasks: Succeeded/Total |
|---|---|---|---|---|
| 0 | collect at <console>:14 +details | 2016/07/06 13:39:51 | 0.2 s | 4/4 |

Lightbend

DATA SCIENCE RETREAT®

# Tasks

### Aggregated Metrics by Executor

| Executor ID ▲ | Address | Task Time | Total Tasks | Failed Tasks | Killed Tasks | Succeeded Tasks |
|---|---|---|---|---|---|---|
| driver | 127.0.1.1:36855 | 0.5 s | 4 | 0 | 0 | 4 |

### Tasks

| Index ▲ | ID | Attempt | Status | Locality Level | Executor ID / Host | Launch Time | Duration | Scheduler Delay | Task Deserialization Time | GC Time | Result Serialization Time | Getting Result Time | Peak Execution Memory | Errors |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 16:04:30 | 36 ms | 99 ms | 43 ms | | 0 ms | 0 ms | 0.0 B | |
| 1 | 1 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 16:04:30 | 17 ms | 42 ms | 45 ms | | 0 ms | 0 ms | 0.0 B | |
| 2 | 2 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 16:04:30 | 29 ms | 28 ms | 49 ms | | 0 ms | 0 ms | 0.0 B | |
| 3 | 3 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 16:04:30 | 12 ms | 53 ms | 48 ms | | 0 ms | 0 ms | 0.0 B | |

Number of tasks equals number of RDD partitions

# Output

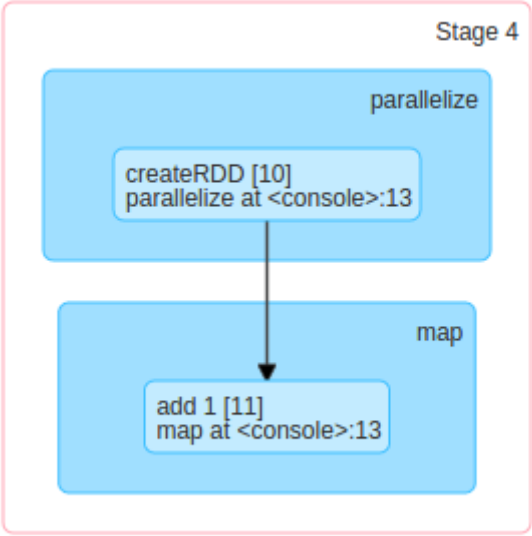
```
adding task set 0.0 with 4 tasks
16/07/06 13:39:51 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (
        TID 0, localhost, partition 0, PROCESS_LOCAL, 5288 bytes)
16/07/06 13:39:51 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (
        TID 1, localhost, partition 1, PROCESS_LOCAL, 5288 bytes)
16/07/06 13:39:51 INFO TaskSetManager: Starting task 2.0 in stage 0.0 (
        TID 2, localhost, partition 2, PROCESS_LOCAL, 5288 bytes)
16/07/06 13:39:51 INFO TaskSetManager: Starting task 3.0 in stage 0.0 (
        TID 3, localhost, partition 3, PROCESS_LOCAL, 5288 bytes)

16/07/06 13:39:51 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)
16/07/06 13:39:51 INFO Executor: Running task 3.0 in stage 0.0 (TID 3)
16/07/06 13:39:51 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
16/07/06 13:39:51 INFO Executor: Running task 2.0 in stage 0.0 (TID 2)
```

Task (**TID**) can be used to find tasks in Spark Web UI

# Naming RDDs

```
val namedTrans =
 sparkSession.sparkContext.
  parallelize(List(1,2,3,4), 4).
   setName(“createRDD").
  map(_ + 1).
   setName(“add 1")
```



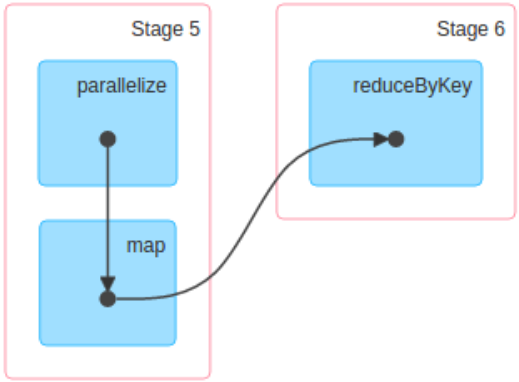

Lightbend

DATA SCIENCE RETREAT®

# Example with Two Stages

```
val sc = sparkSession.sparkContext
val transformation2Stages =
  //stage 1
  sc.parallelize(List(1,2,3,4,1,2,3,4),4).
        setName("createRDD").
      map(value => (value, 1)).
        setName("makeKeyValue").
  //stage 2
      reduceByKey(_+_).
        setName("reduceSum")

transformation2Stages.toDebugString
```




```
(4) reduceSum ShuffledRDD[6]
+-(4) makeKeyValue MapPartitionsRDD[5]
  | createRDD ParallelCollectionRDD[4]
```

Denotes a stage

Lightbend

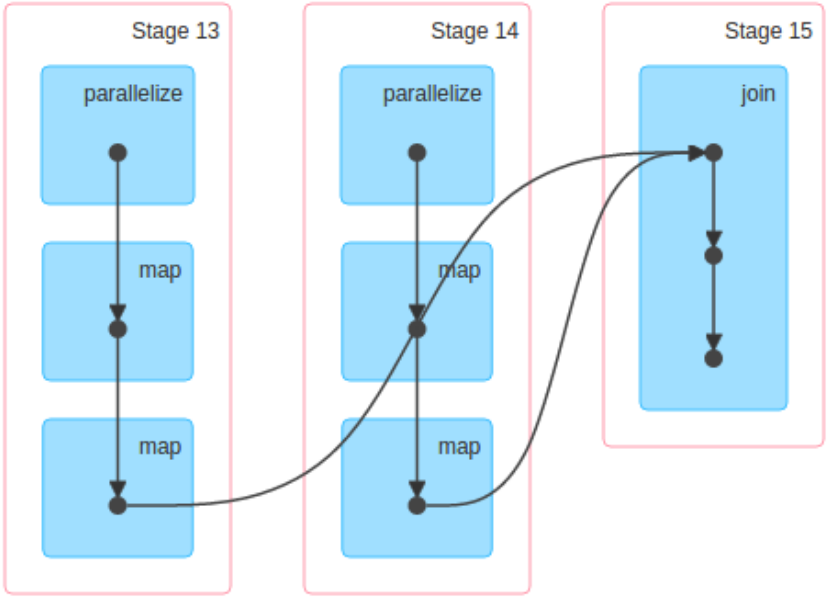
DATA SCIENCE RETREAT®

# Repeated Computation: No Cache

```
val initialRDD =
   sc.parallelize(List(1,2,3,4), numSlices = 4).
      setName("createRDD")

val slowComputation = initialRDD.map(key => {
   Thread.sleep(5000); key}). /*cache()*/
   setName("slowComputation")

val remap1 = slowComputation.map(k => (k, 1)).
   setName("remap_1")
val remap2 = slowComputation.map(k => (k + 1, 1)).
    setName("remap_2")
val joined = remap1.join(remap2).setName("joined")
joined.count()
```

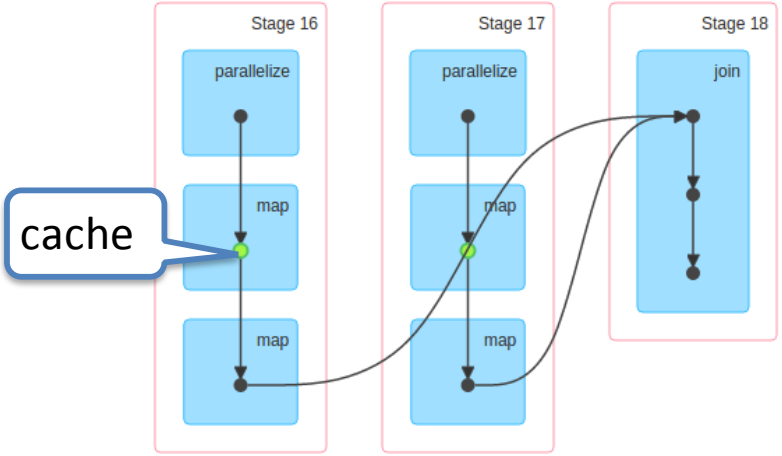


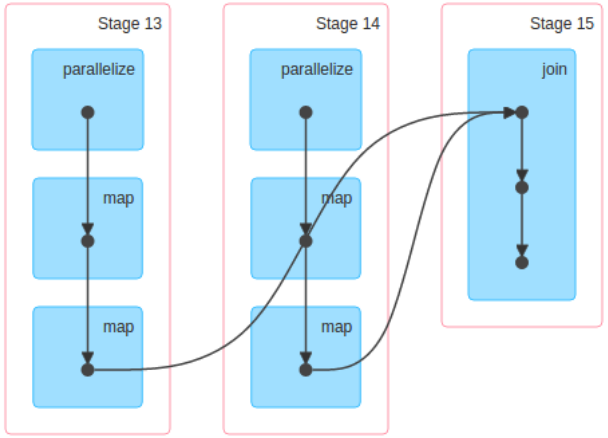

Lightbend

DATA SCIENCE RETREAT®

# Cache vs No Cache

```
val slowComputation = initialRDD.map(key => {
  Thread.sleep(5000); key}).cache()
  setName("slowComputation")
```




Lightbend

DATA SCIENCE RETREAT®

# Effect of Cache

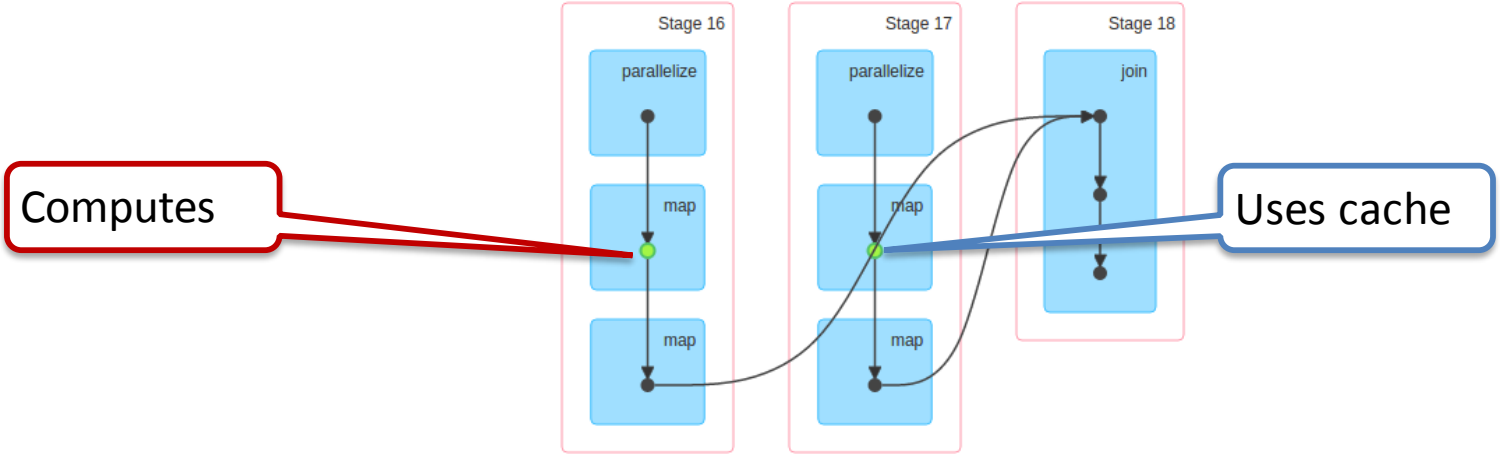
Cache

**Completed Jobs (11)**

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 10 | count at <console>:19 | 2016/07/06 17:23:22 | 5 s | 3/3 | 12/12 |
| 9 | count at <console>:30 | 2016/07/06 17:21:37 | 10 s | 3/3 | 12/12 |

No Cache

# Computation vs. Cache




| Index ▲ | ID | Attempt | Status | Locality Level | Executor ID / Host | Launch Time | Duration |
|---|---|---|---|---|---|---|---|
| 0 | 56 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:22 | 5 s |
| 1 | 57 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:22 | 5 s |
| 2 | 58 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:22 | 5 s |
| 3 | 59 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:22 | 5 s |

| Index ▲ | ID | Attempt | Status | Locality Level | Executor ID / Host | Launch Time | Duration |
|---|---|---|---|---|---|---|---|
| 0 | 60 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:27 | 16 ms |
| 1 | 61 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:27 | 2 ms |
| 2 | 62 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:27 | 1 ms |
| 3 | 63 | 0 | SUCCESS | PROCESS_LOCAL | driver / localhost | 2016/07/06 17:23:27 | 2 ms |

# Storage Tab

```
val slowComputation = initialRDD.map(key => {
  Thread.sleep(5000); key}).cache()
  setName("slowComputation")
```

Spark 2.0.0-SNAPSHOT | Jobs | Stages | Storage | Environment | Executors | SQL | My Spark Application application UI

## Storage

### RDDs

| RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size on Disk |
|---|---|---|---|---|---|
| slowComputation | Memory Deserialized 1x Replicated | 4 | 100% | 96.0 B | 0.0 B |

Note: the effect of cache on performance depends on the available memory in Spark and on other jobs running and using the cache in the same Spark application. Consider using persist with a disk serialization option.

Lightbend

DATA SCIENCE RETREAT®

# Lesson Summary

- Having completed this lesson, you should be able to:
  - Understand how concepts like cores, executors, tasks and stages apply to actual Spark applications
  - Be able to use Spark Web UI to understand how your Spark application works
  - Be able to find the running time of each job, stage, and task