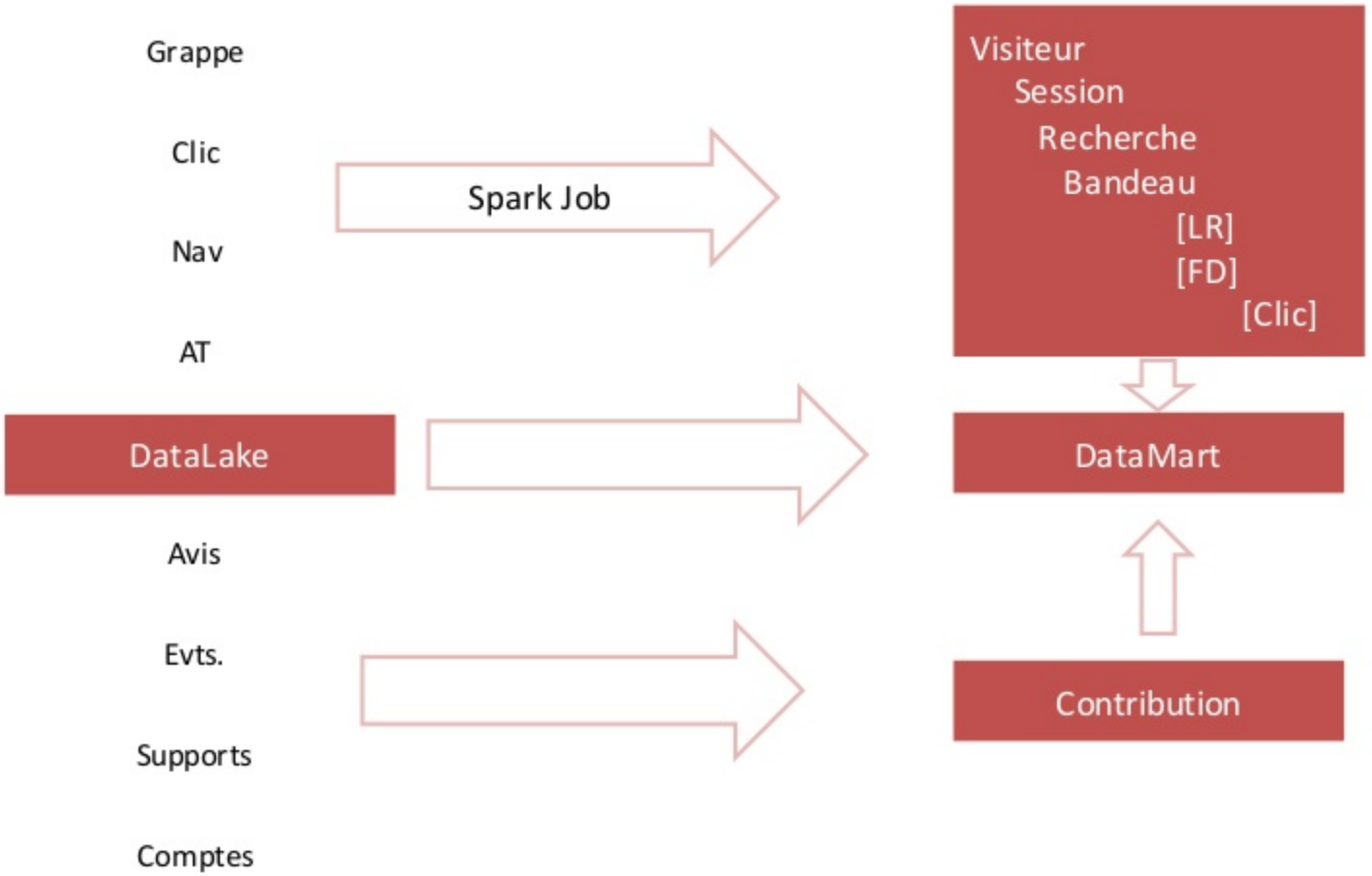
ebiznext

HAYSSAM SALEH

Spark / Mesos Cluster Optimization

Paris Spark Meetup April 28th 2016 @Criteo

Project Context



Interactive Discovery

Elastcsearch DataMart
(KPI)

ebiznext

Interactive Discovery

Coming as a Kibana 5 plugin

		buyerCountry	FR			
		buyerCivility	MR		MRS	
		Metric(s)	count (6)	sum (totalPrice)	count (21)	sum (totalPrice)	Total
productCategoryName	productName						
Exercise Mats	Premium Yoga Pink Grey Mat				100.0%	20000	4
	Yoga Black Solar Blue Mat		50.0%	4500	50.0%	4500	2
Fitness Leggings and Pants	Pink Womens Slim Fitness Pants				100.0%	3000	1
Fitness T Shirts	Womens White Gym T-Shirt		100.0%	900			1
Football Boots	Superteam Football Boots				100.0%	47996	4
	Vortex Football Boots		14.29%	14900	85.71%	104300	7
Football Shirts desc	Electric Blue Womens Football Shirt		100.0%	1998			2
Football Socks	Pale Blue and Black Football Socks				100.0%	345	1
Footballs	Champion Blue Football				100.0%	1600	1
	England Football				100.0%	1398	2
Golf Polo Shirts desc	Pale Pink Womens Polo Shirt		100.0%	0			1
Tennis Rackets	Modern Wood Tennis Racket				100.0%	5000	1

Elastcsearch DataMart
(KPI)

OBJECTIVES

- 100 Gb of data / day => 50 million requests
- 40To / year

- How we turned from a 4 hours job on :
  - 6 nodes
  - 8 cores & 32 Gb per node

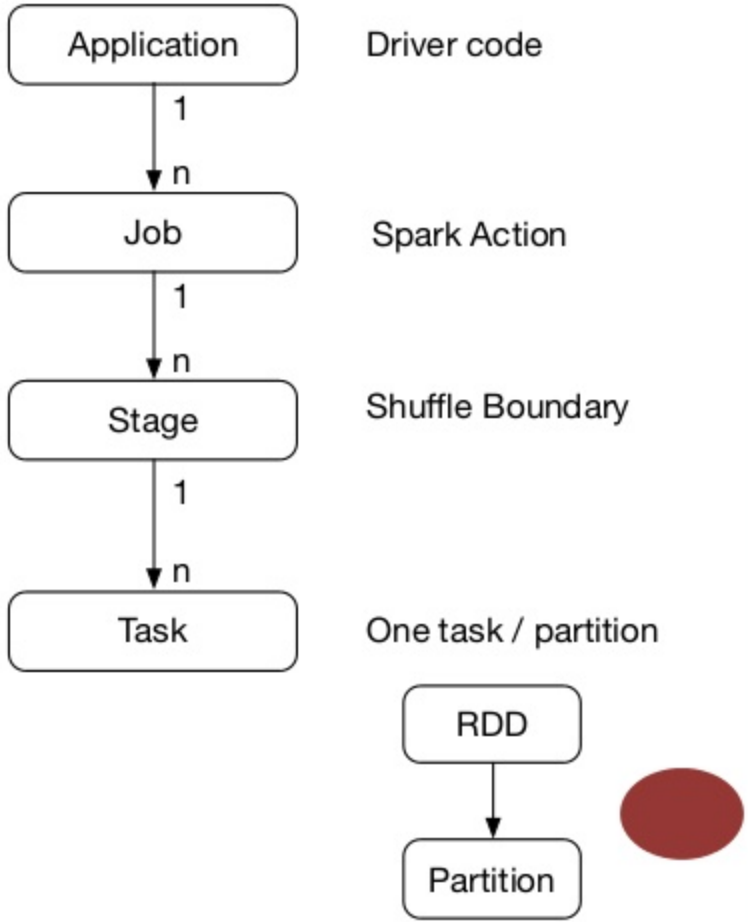
  - To a 20 minutes Job on :
  - 4 nodes, 8 cores & 8 Gb per node

SUMMARY

- Spark concepts
- Spark UI offline
- Application Optimization
  - Shuffling
  - Partitioning
  - Closures
- Parameters Optimization
  - Spark Shuffling
  - Mesos application distribution
- Elasticsearch Optimization
  - Google “elasticsearch performance tuning” -> blog.ebiznext.com

SPARK : LES CONCEPTS

- Application
  - Main application
- Job
  - Roundtrip Driver -> Cluster
- Stage
  - Shuffle Boundary
- Task
  - Thread working on a single RDD partition
- Partition
  - RDD are split into partitions
  - Partition is unit of work for each task
- Executor
  - System process



SPARK UI OFFLINE

On the driver

```
spark {
  master = "mesos://zk://app200.cluster1:2181/mesos"
  local.dir=/home/tmp
  executor.extraClassPath="/home/pj/datalab/datacrunch/lib/*"
  executor.uri="http://download.ebiznext.com/spark-1.6.0-bin-hadoop2.6.tgz"
  app.id=datacrunch
  mesos.executor.home=/home/pj/datalab/spark
  ui.enabled=true
  executor.memory=16g
  driver.memory=4g
  eventLog.enabled=true
  eventLog.dir=/tmp/spark-events
  eventLog.compress=true
}
```

Spark-env.sh

```
export SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=file:/opt/spark/spark-events -Dspark.history.fs.cleaner.enabled=true"
```

```
./spark-1.6.0-bin-hadoop2.6/sbin/start-history-server.sh
```

APPLICATION

Spark 1.6.0 **History Server**

Event log directory: file:/Users/hayssams/programs/spark-1.6.0-bin-hadoop2.6/spark-events/

Showing 1-20 of 20 1

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0067	Datalab	2016/04/25 20:54:39	2016/04/25 21:46:06	51 min	root	2016/04/25 21:46:06
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0066	Datalab	2016/04/25 19:56:51	2016/04/25 20:54:38	58 min	root	2016/04/25 20:54:38
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0065	Datalab	2016/04/25 19:10:28	2016/04/25 19:56:50	46 min	root	2016/04/25 19:56:50
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0064	Datalab	2016/04/25 18:35:32	2016/04/25 19:10:26	35 min	root	2016/04/25 19:10:26
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0062	Datalab	2016/04/25 17:11:42	2016/04/25 17:52:42	41 min	root	2016/04/25 17:52:42
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0061	Datalab	2016/04/25 16:02:40	2016/04/25 16:25:50	23 min	root	2016/04/25 16:25:50
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0060	Datalab	2016/04/25 16:00:55	2016/04/25 16:01:17	23 s	root	2016/04/25 16:01:17
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-0059	Datalab	2016/04/25 15:59:23	2016/04/25 15:59:46	23 s	root	2016/04/25 15:59:46

JOBS

Total Uptime: 51 min
Scheduling Mode: FIFO
Completed Jobs: 70

▸ Event Timeline

Completed Jobs (70)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
69	collect at AudienceVariation.scala:76	2016/04/25 21:45:37	29 s	2/2	2048/2048
68	foreach at AudienceVariation.scala:54	2016/04/25 21:44:24	1.2 min	1/1	1024/1024
67	countByValue at AudienceVariation.scala:38	2016/04/25 21:37:00	7.2 min	2/2	2048/2048
66	parquet at AudienceVariation.scala:34	2016/04/25 21:36:59	1 s	1/1	1024/1024
65	collect at GReferentielLoader.scala:72	2016/04/25 21:36:57	0.4 s	1/1	2/2
64	collect at GReferentielLoader.scala:53	2016/04/25 21:36:56	0.3 s	1/1	1/1
63	collect at GReferentielLoader.scala:90	2016/04/25 21:36:56	62 ms	1/1	2/2
62	take at CsvRelation.scala:174	2016/04/25 21:36:56	23 ms	1/1	1/1
61	take at CsvRelation.scala:174	2016/04/25 21:36:56	22 ms	1/1	1/1
60	take at CsvRelation.scala:174	2016/04/25 21:36:56	37 ms	1/1	1/1
59	collect at AudienceVariation.scala:76	2016/04/25 21:36:47	9 s	2/2	2048/2048
58	foreach at AudienceVariation.scala:54	2016/04/25 21:36:03	44 s	1/1	1024/1024
57	countByValue at AudienceVariation.scala:38	2016/04/25 21:31:49	4.2 min	2/2	2048/2048
56	parquet at AudienceVariation.scala:34	2016/04/25 21:31:47	1 s	1/1	1024/1024
55	collect at GReferentielLoader.scala:72	2016/04/25 21:31:45	0.4 s	1/1	2/2
54	collect at GReferentielLoader.scala:53	2016/04/25 21:31:45	0.3 s	1/1	1/1

STAGES

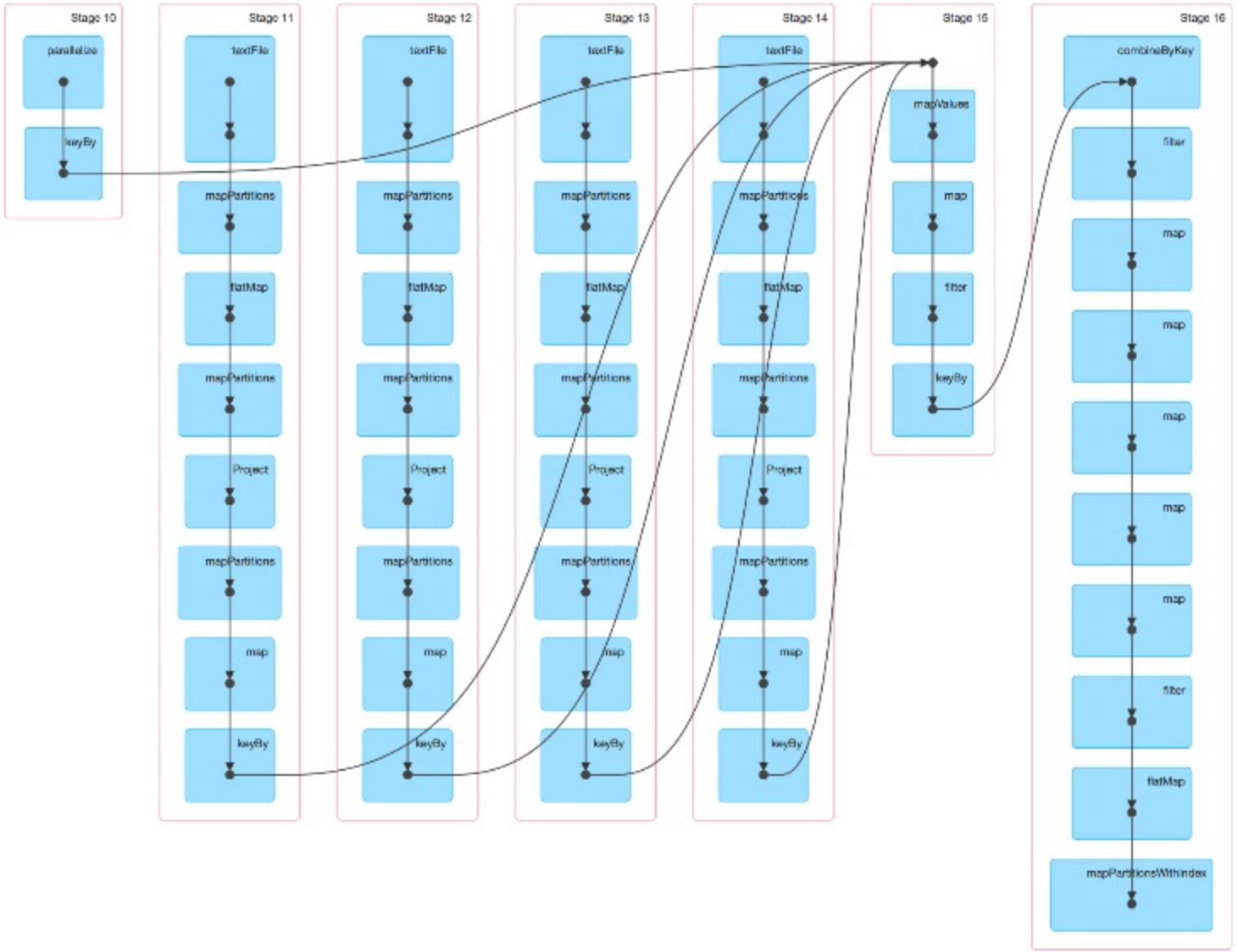
Status: SUCCEEDED
Completed Stages: 7

- Event Timeline
- DAG Visualization

Completed Stages (7)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
16	reduce at LoadES.scala:116	+details	2016/04/25 10:42:20	4.5 min	1024/1024			10.7 GB	
15	keyBy at CoGroupFixe.scala:85	+details	2016/04/25 10:34:52	7.5 min	1024/1024			10.2 GB	10.7 GB
14	keyBy at CoGroupFixe.scala:78	+details	2016/04/25 10:30:16	4.6 min	50/50	6.1 GB			9.7 GB
13	keyBy at CoGroupFixe.scala:80	+details	2016/04/25 10:30:16	22 s	1/1	8.7 MB			19.2 MB
12	keyBy at CoGroupFixe.scala:81	+details	2016/04/25 10:30:16	29 s	8/8	252.5 MB			533.4 MB
11	keyBy at CoGroupFixe.scala:81	+details	2016/04/25 10:30:16	2 s	1/1	148.0 B			
10	keyBy at CoGroupFixe.scala:81	+details	2016/04/25 10:30:16	2 s	1024/1024				

STAGES


Stage 10
parallelize
keyBy
Stage 11
textFile
mapPartitions
flatMap
mapPartitions
Project
mapPartitions
map
keyBy
Stage 12
textFile
mapPartitions
flatMap
mapPartitions
Project
mapPartitions
map
keyBy
Stage 13
textFile
mapPartitions
flatMap
mapPartitions
Project
mapPartitions
map
keyBy
Stage 14
textFile
mapPartitions
flatMap
mapPartitions
Project
mapPartitions
map
keyBy
Stage 15
mapValues
map
filter
keyBy
Stage 16
combineByKey
filter
map
map
map
map
map
filter
flatMap
mapPartitionsWithIndex

TASKS

Summary Metrics for 1024 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	1 ms	2 ms	3 ms	6 ms	0.1 s
GC Time	0 ms	0 ms	0 ms	0 ms	94 ms

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S0/3	app230.cluster1:44817	46 s	408	0	408
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S1/2	app227.cluster1:35871	49 s	447	0	447
be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S2/1	dat229.cluster1:47180	29 s	169	0	169

Tasks

Page: 1 2 3 4 5 6 7 8 9 10 > >>

11 Pages. Jump to 1 . Show 100 items in a page. Go

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Errors
0	12	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S2/1 / dat229.cluster1	2016/04/25 10:30:16	0.1 s	94 ms	
1	13	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S1/2 / app227.cluster1	2016/04/25 10:30:16	54 ms		
2	14	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S0/3 / app230.cluster1	2016/04/25 10:30:16	79 ms		
3	15	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S2/1 / dat229.cluster1	2016/04/25 10:30:16	0.1 s	94 ms	
4	16	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S1/2 / app227.cluster1	2016/04/25 10:30:16	67 ms		
5	17	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S0/3 / app230.cluster1	2016/04/25 10:30:16	83 ms		
6	18	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S2/1 / dat229.cluster1	2016/04/25 10:30:16	0.1 s	94 ms	
7	19	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S1/2 / app227.cluster1	2016/04/25 10:30:16	70 ms		
8	20	0	SUCCESS	PROCESS_LOCAL	be453b87-3573-4cb8-8ab0-c61bd0e4a8a8-S0/3 / app230.cluster1	2016/04/25 10:30:16	73 ms		

Shuffling Application Optimization

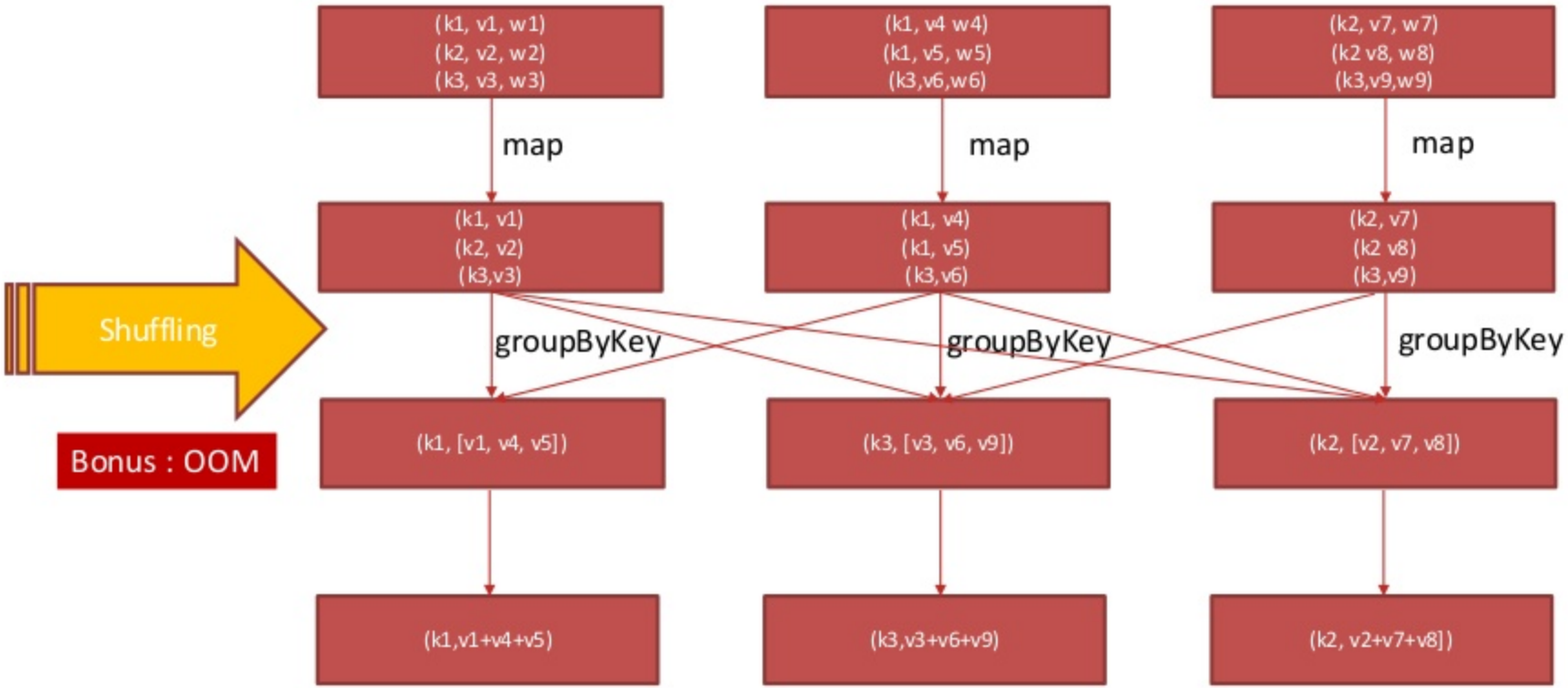
Why Optimize Shuffling

Distance between Data & CPU	Duaration (scaled)
Cache L1	1 seconde
RAM	3 minutes
Node to node communication	3 jours

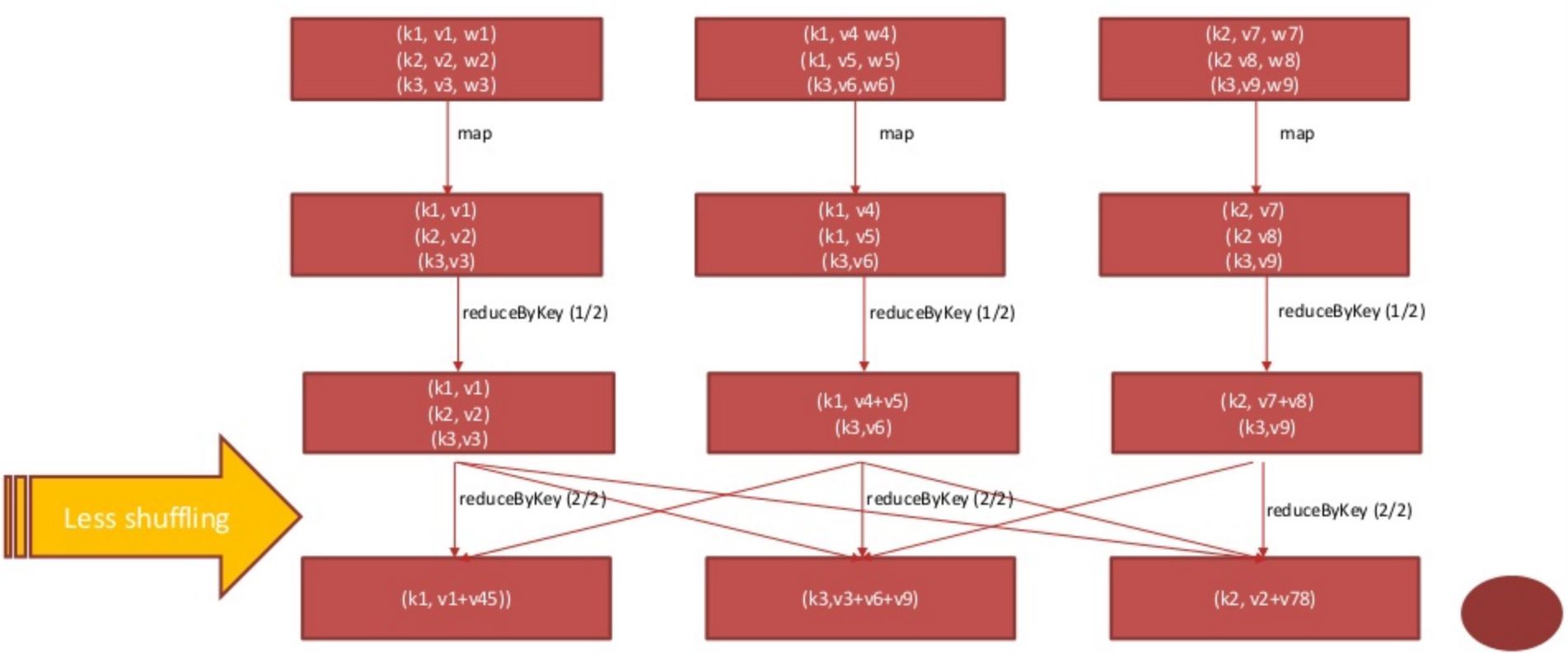
TRANSFORMATIONS LEADING TO SHUFFLING

- `repartition`
- `cogroup`
- `...join`
- `...ByKey`
- `distinct`

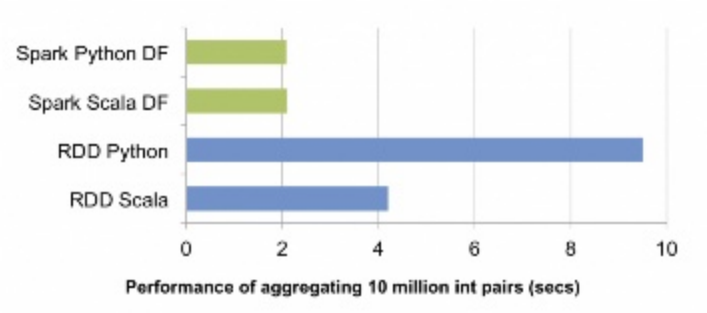
SHUFFLING OPTIMIZATION 1/2



SHUFFLING OPTIMIZATION 2/2



Optimizations brought by Spark SQL



☹ Weak typing

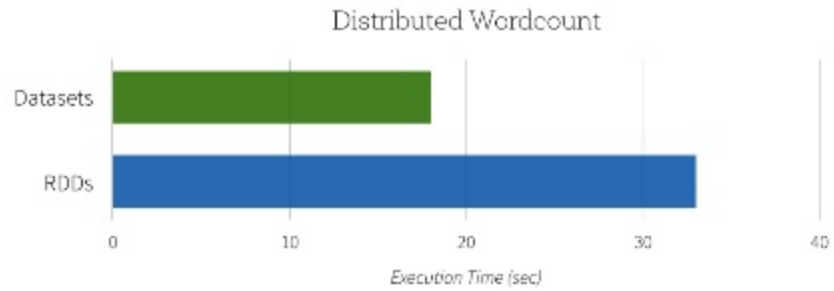
```
sqlContext.sql(s"select $a, SUM(_2) from datamart group by $a").show(1000)
```

☹ Limited Java Support

☹ Schema inference not supported

- requires scala.Product

LES DATASETS – EXPERIMENTAL -



☺ API similar to RDD, strongly typed

```
dataset.filter(_.age < 21)
```

```
dataset.filter(person -> person.getAge() < 21);
```

REDUCE OBJECT SIZE

CLOSURES

```
class MySparkApp extends Serializable {
  val ref1 : Map[String, ESRecherche] =
  val ref2 : Map[String, ESRecherche] =
  val ref3 : Map[String, ESRecherche] =
  val ref4 : Map[String, ESRecherche] =
  val ref5 : Map[String, ESRecherche] =
  def doIt() :Unit = {
    myRDD.map(x => ref3.get(key))
  }
}
```

```
myRDD.map(x =>      ref3.get(key))
```

CLOSURES

```
class MySparkApp extends Serializable {
  val ref1 : Map[String, ESRecherche] =
  val ref2 : Map[String, ESRecherche] =
  val ref3 : Map[String, ESRecherche] =
  val ref4 : Map[String, ESRecherche] =
  val ref5 : Map[String, ESRecherche] =
  def doIt() :Unit = {
    myRDD.map(x => ref3.get(key))
  }
}
```

Indirect reference. Use local variables instead

```
myRDD.map(x => this.ref3.get(key))
```

RIGHT PARTITIONING

PARTITIONING

- Symptoms
  - Important number of short lived tasks
- Solutions
  - Review your implementation
  - Define custom partitionner
  - Control the number of partitions with
    - coalesce et repartition

Duration	GC Time	Shuffle Read Size / Records
25 ms		0.0 B / 0
58 ms		0.0 B / 0
39 ms		0.0 B / 0
27 ms		0.0 B / 0
58 ms		0.0 B / 0
41 ms		0.0 B / 0
27 ms		0.0 B / 0
62 ms		0.0 B / 0
43 ms		0.0 B / 0
27 ms		0.0 B / 0
65 ms		0.0 B / 0

CHOOSE THE RIGHT SERIALIZATION ALGORITHM

REDUCE OBJECT SIZE

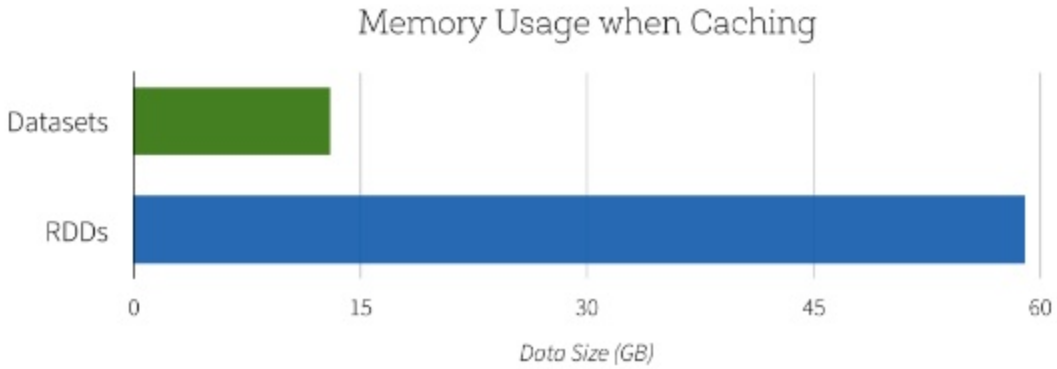
- Kryo Serialization
  - Reduce space – up to 10X
  - Perform gain – 2X to 3X

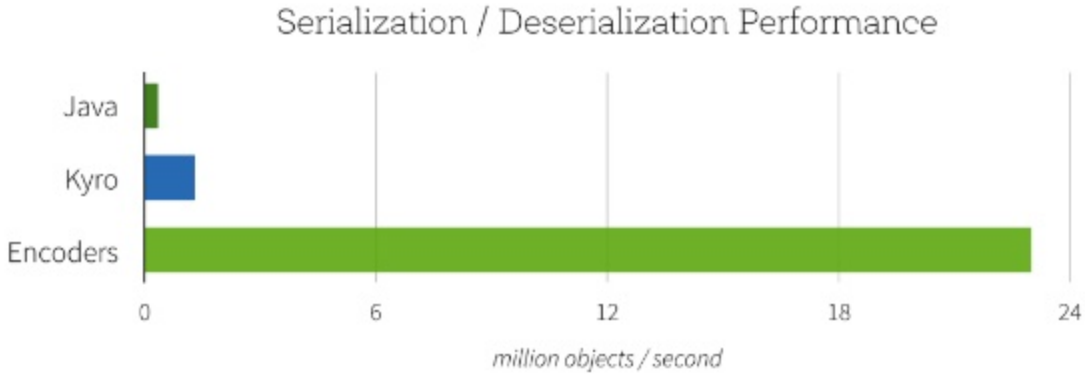
```
spark.serializer = "org.apache.spark.serializer.KryoSerializer"
spark.kryo.classesToRegister = ... ma liste de classes à sérialiser
spark.kryo.registrationRequired = true
```

1. Require Spark to use Kryo serialization
2. Optimize class naming => reduce object size
3. Force all serialized classes to register

Not applicable to the Dataset API that use Encoders instead

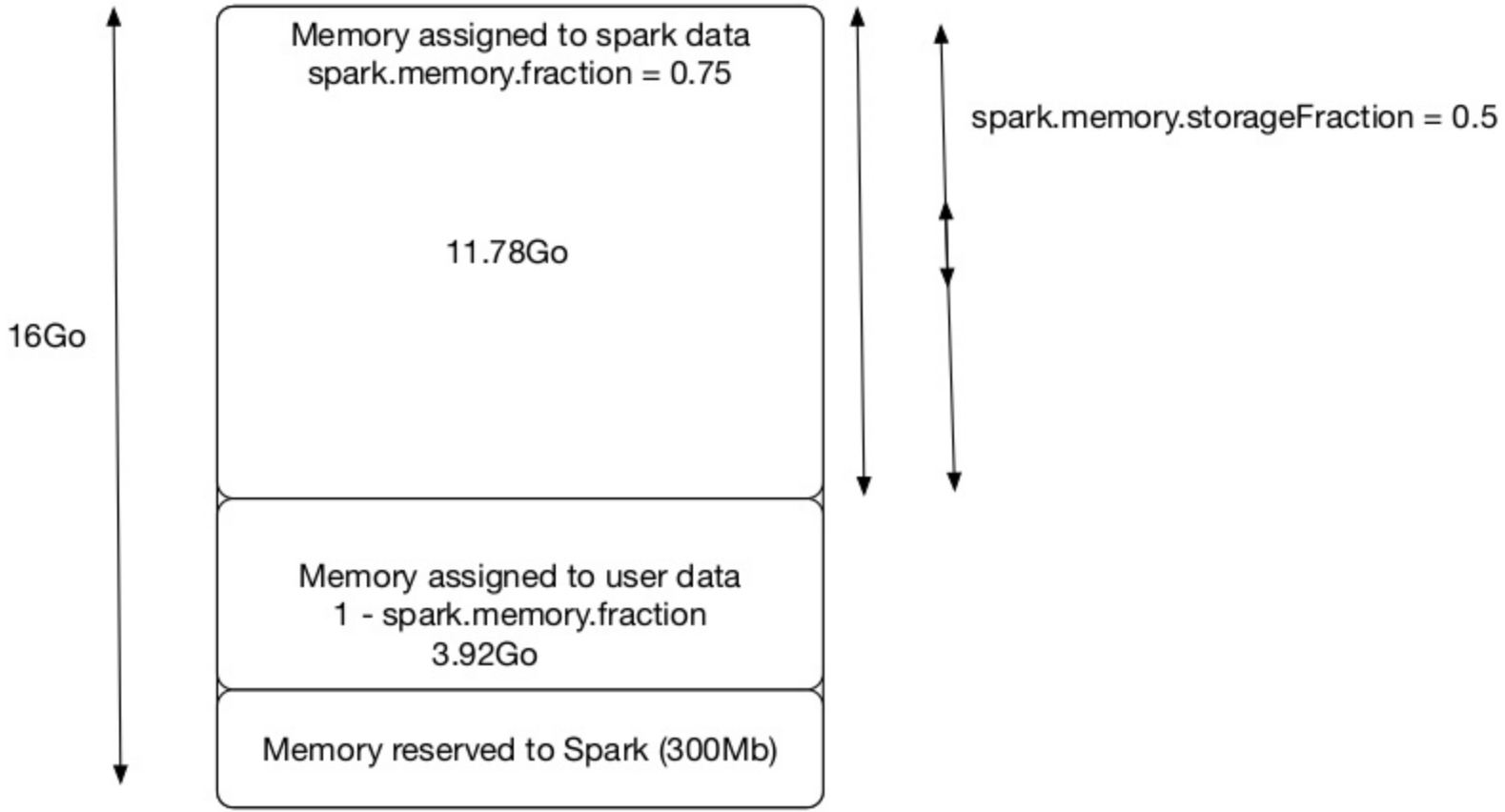
DATASETS PROMISE





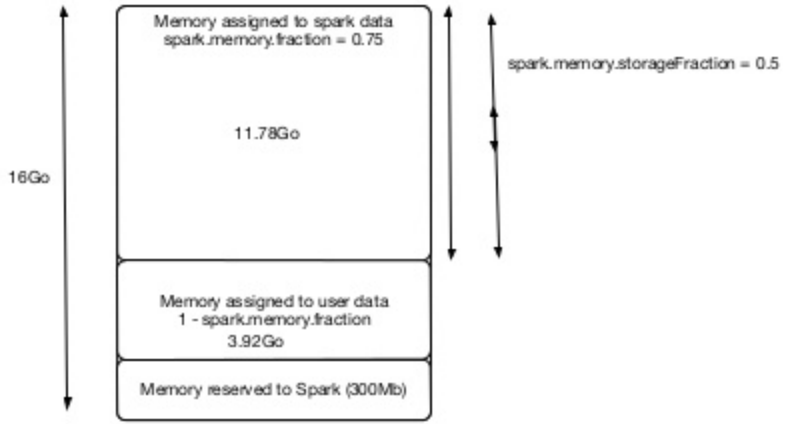
MEMORY TUNING

SPARK MEMORY MANAGEMENT– SPARK 1.6.X



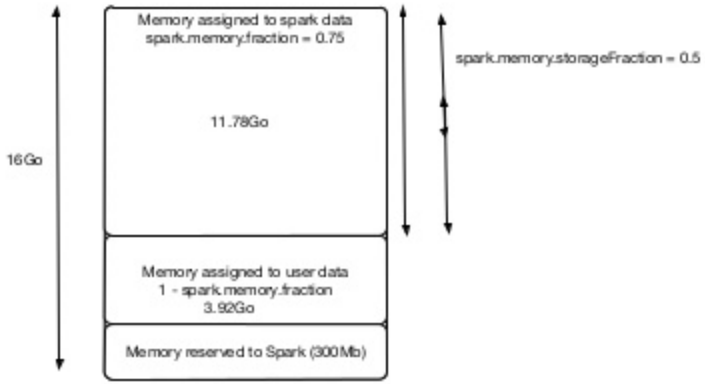
Spark Memory Management – Spark 1.6.X

- Storage Fraction
  - Host cached RDDs
  - Host broadcast variables
  - Data in this fraction are subject to eviction
- Shuffling Fraction
  - Hold Intermediate Data
  - Spill to disk when full
  - Data this fraction cannot be evicted by other threads



SPARK MEMORY MANAGEMENT – SPARK 1.6.X

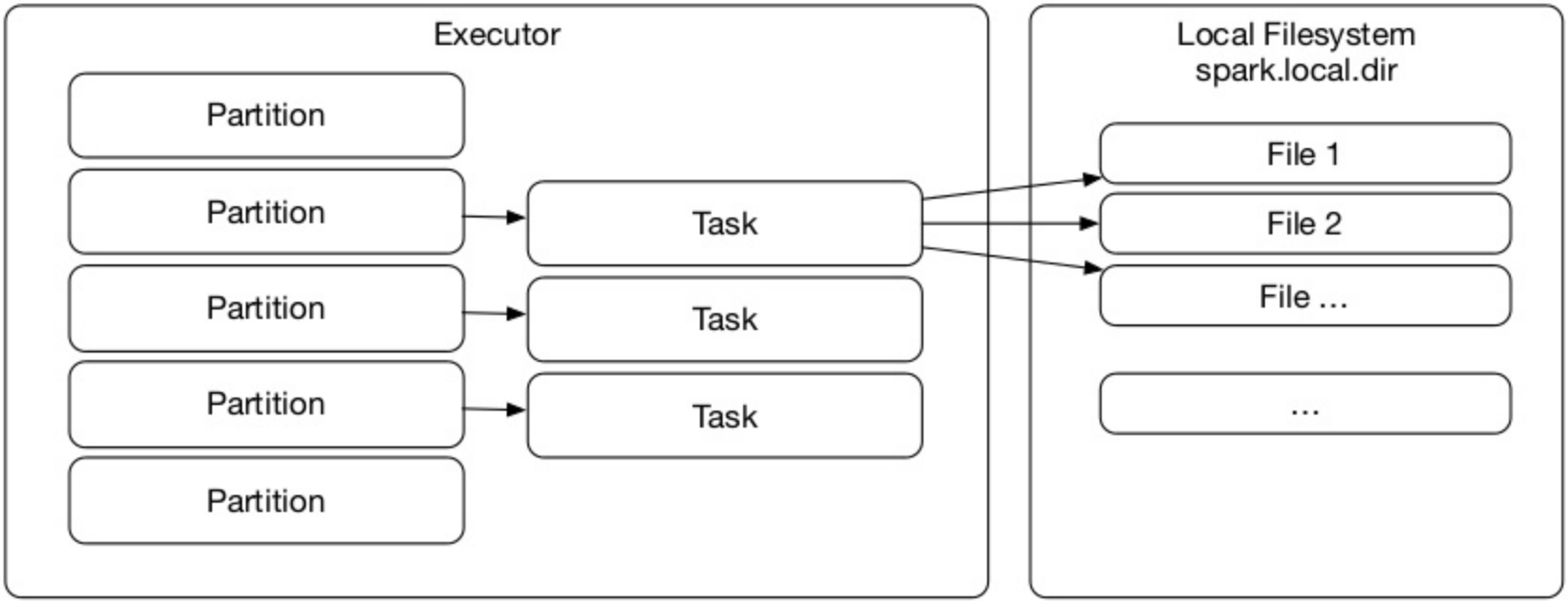
- Shuffle & Storage fractions may borrow memory from each other under certain conditions:
  - The shuffling fraction cannot extend beyond its defined size if storage fraction uses all its memory
  - Storage fraction cannot evict data from the shuffling fraction even if it expanded beyond its defined size.



THE SHUFFLE MANAGER

SPARK.SHUFFLE.MANAGER

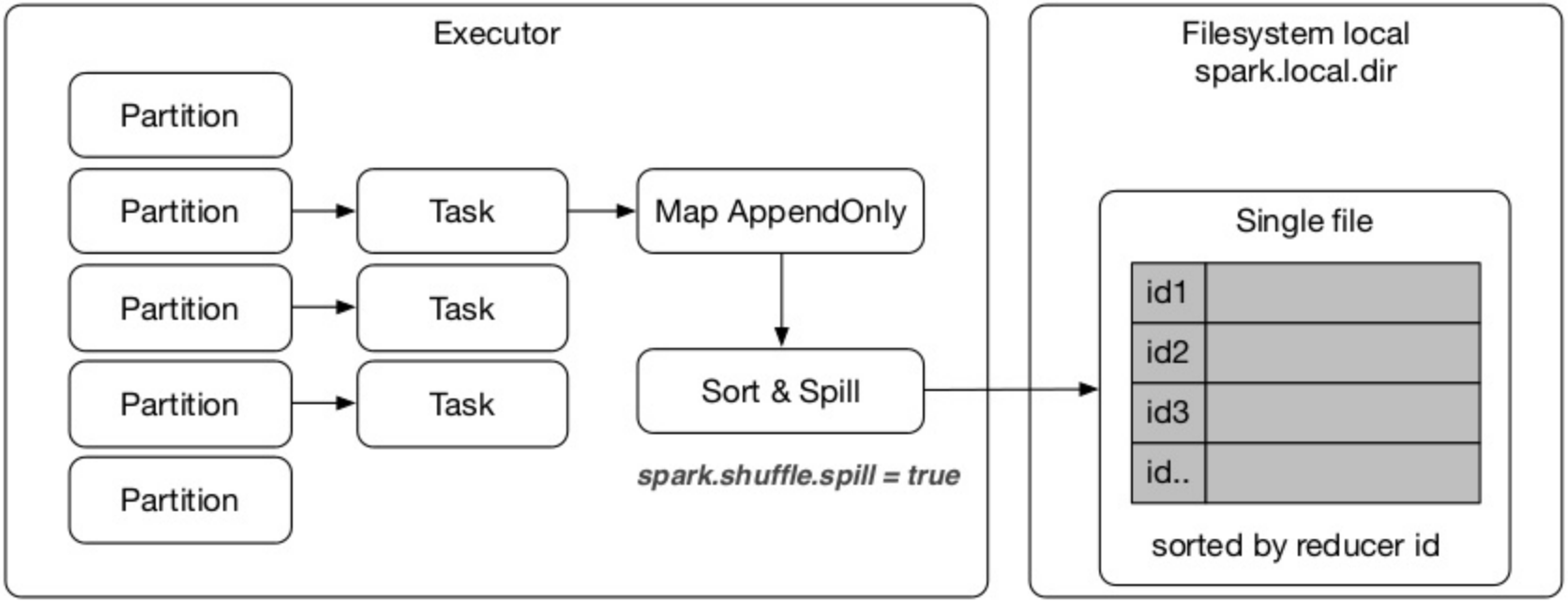
HASH SHUFFLE MANAGER



Number of tasks = spark.executor.cores / spark.task.cpus(1)

YARN & standalone only

SORT SHUFFLE MANAGER



Number of tasks = spark.executor.cores / spark.task.cpus(1)

YARN & standalone only

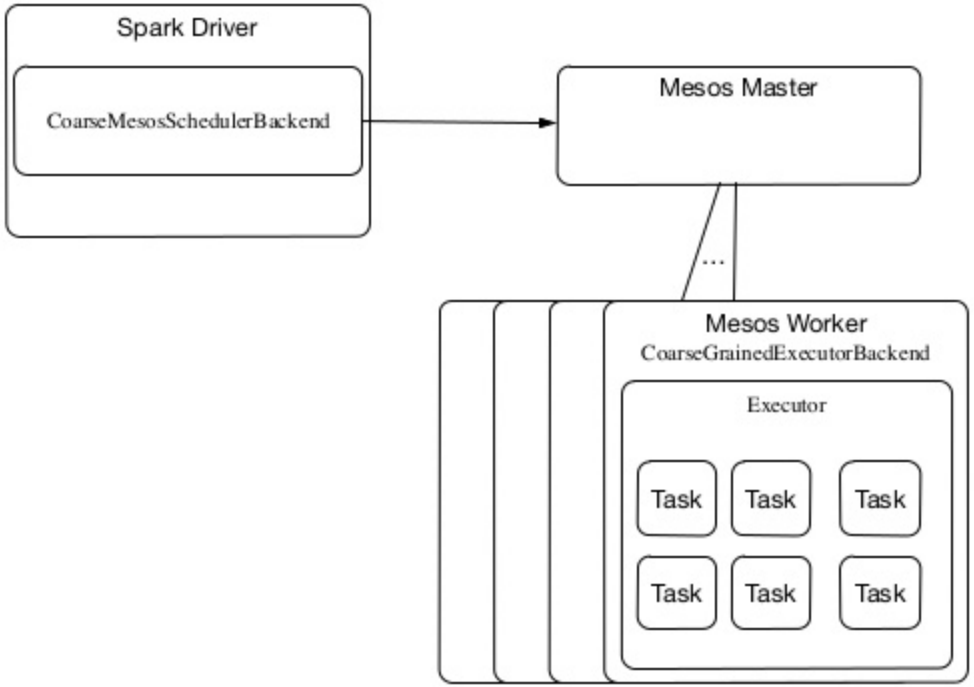
SHUFFLE MANAGER

- ***spark.shuffle.manager*** = *hash*
  - Perform better when the number of mapper/reducer is small (generation of M * R files)
- ***spark.shuffle.manager*** = *sort*
  - Perform better for an important number of mapper/reducer
- Best of both world
  - **spark.shuffle.sort.bypassMergeThreshold**
- ***spark.shuffle.manager*** = tungsten-sort ???
  - Similar to sort with the following benefits :
    - Off-Heap allocation
    - Work directly on binary object serialization (much smaller than JVM objects) – aka memcopy ☺
    - Use 8 bytes / record pour les sort => take advantage on L* cache

Spark on Mesos

COARSE GRAINED MODE

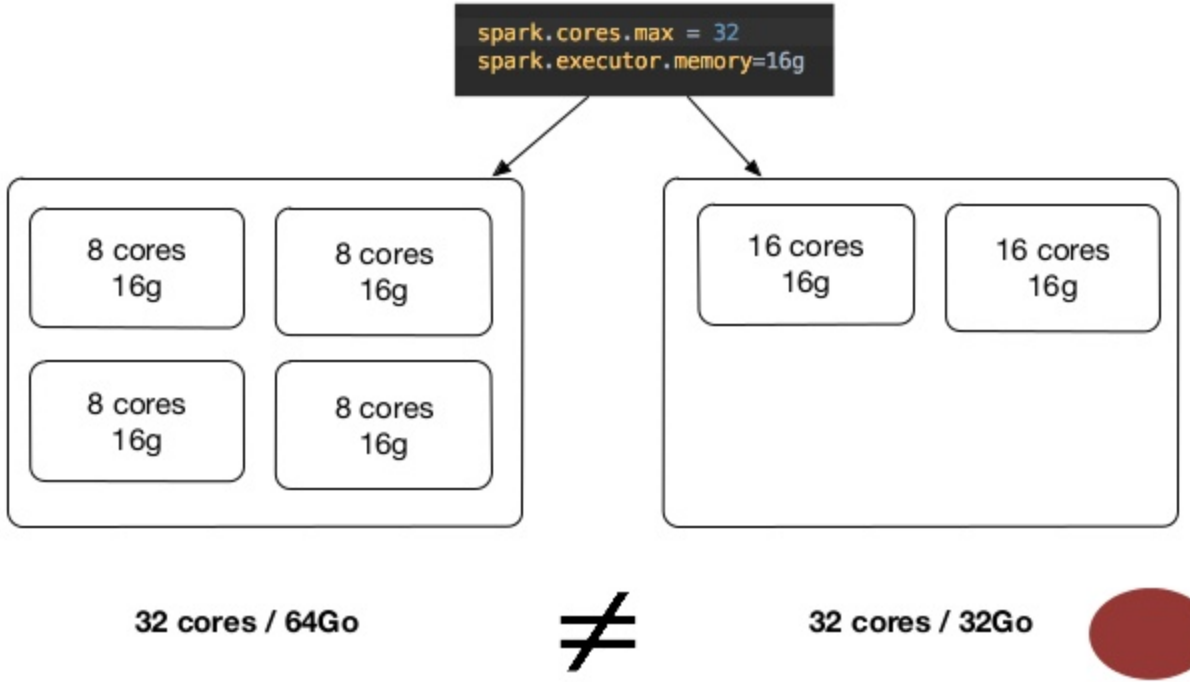
☺ Static resource allocation



COARSE GRAINED MODE

☹ Only one executor / node

☹ No control over the number of executors

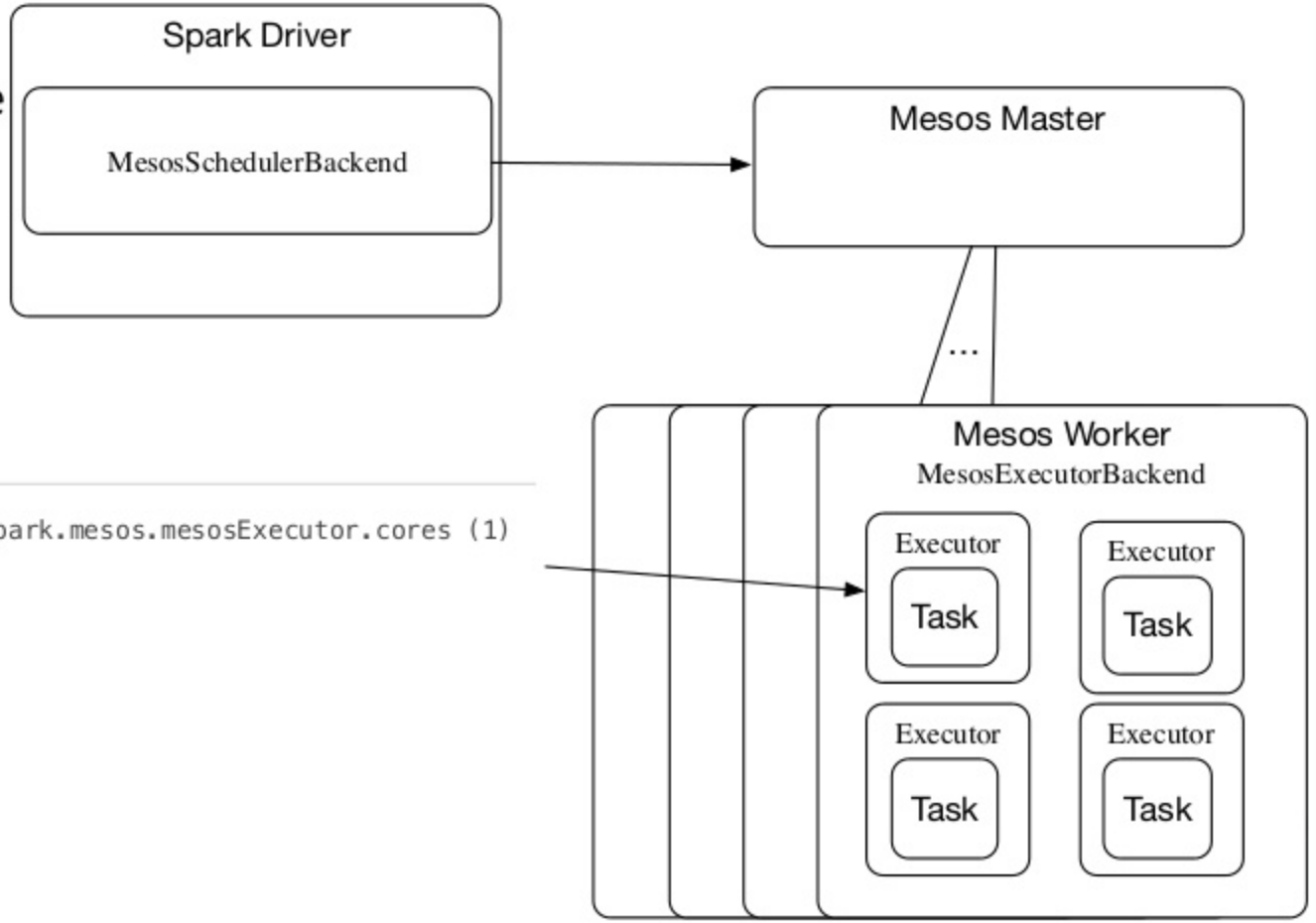


Fine Grained Mode

☹ Only one executor / node

☹ No guaranty of resource availability

☹ No control over the number of executors




```
spark.mesos.mesosExecutor.cores (1)
```

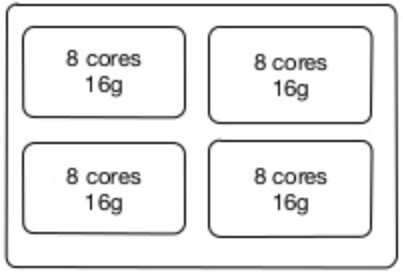
Solution 1 : Use Mesos Roles

- Limit the number of cores / node on each mesos worker

```
mesos-slave --resources="cpus(sparkapp):8,mem(sparkapp):16,cpus():32,mem():48"
```

- Assign a Mesos role to your Spark job

```
spark.cores.max = 32
spark.executor.memory=16g
spark.mesos.role = sparkapp
```



☹ Must be configured for each Spark application

Solution 2 : Dynamic Allocation (Coarsed Grained Mode only)

- Principles
  - Dynamically add/remove Spark executors
- Requires a dedicated process to move shuffled data (spawned on each Mesos worker through Marathon)

```
spark.dynamicAllocation.enabled=true
spark.dynamicAllocation.initialExecutors = 4
spark.dynamicAllocation.minExecutors = 4
spark.dynamicAllocation.maxExecutors = 4
spark.dynamicAllocation.executorIdleTimeout=60s
spark.dynamicAllocation.schedulerBacklogTimeout=1s
```

QUESTIONS ?