# Introduction to Spark Internals

Neil Gibbons

# What makes it difficult to understand Spark intuitively?

- So much functionality (Streaming, Interactive, Batch, Machine Learning Workloads)
- Lack of learning resources on internals
- Huge codebase

# Outline

1. Breaking down computation tasks with Map, Shuffle, Reduce
2. Intuition behind RDDs (recomputation for efficient fault tolerance)
3. Spark Scheduler

# Outline

1. **Breaking down computation tasks with Map, Shuffle, Reduce**
2. Intuition behind RDDs (recomputation for efficient fault tolerance)
3. Spark Scheduler

# Map, Shuffle and Reduce

- If you want to break a computation task down in a way that is parallelisable, this is a key method
- Goes beyond computers
  - Census
  - Creating a book index
- 1890s Hollerith Machine for US Census
- Google's MapReduce
- Core idea behind Spark
  - Builds on MapReduce
  - Not clear that Spark does Map and Reduce (this is hidden behind the scheduler and the transformation methods of RDDs)
  - Just another way of breaking down a computation task, specified by the user, into Map and Reduce stages

# Outline

1. Breaking down computation tasks with Map, Shuffle, Reduce
2. **Intuition behind RDDs (recomputation for efficient fault tolerance)**
    a. **Where the idea comes from**
    b. How they are implemented
3. Spark Scheduler

# Limitations of MapReduce

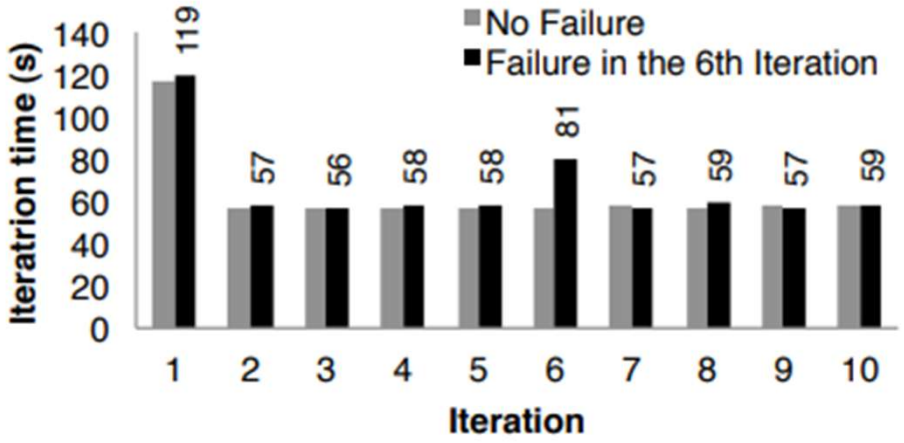
- Batch vs Iterative and Interactive Workloads

# What caused the slowness?

- Looking at the system, where does it spend most of its time?
- Replication for fault tolerance
- Fault tolerance vs Performance

# How else to provide fault tolerance?

- Recomputation? Isn't this a bad idea?
  - Storing computation, rather than materialising data
  - Similar speed in practice, plus, faults not needing to recompute on every iteration
  - 10 to 100 time speed ups for interactive/iterative workloads



# Outline

1. Breaking down computation tasks with Map, Shuffle, Reduce
2. **Intuition behind RDDs (recomputation for efficient fault tolerance)**
    a. Where the idea comes from
    **b. How they are implemented**
3. Spark Scheduler

# RDD Class

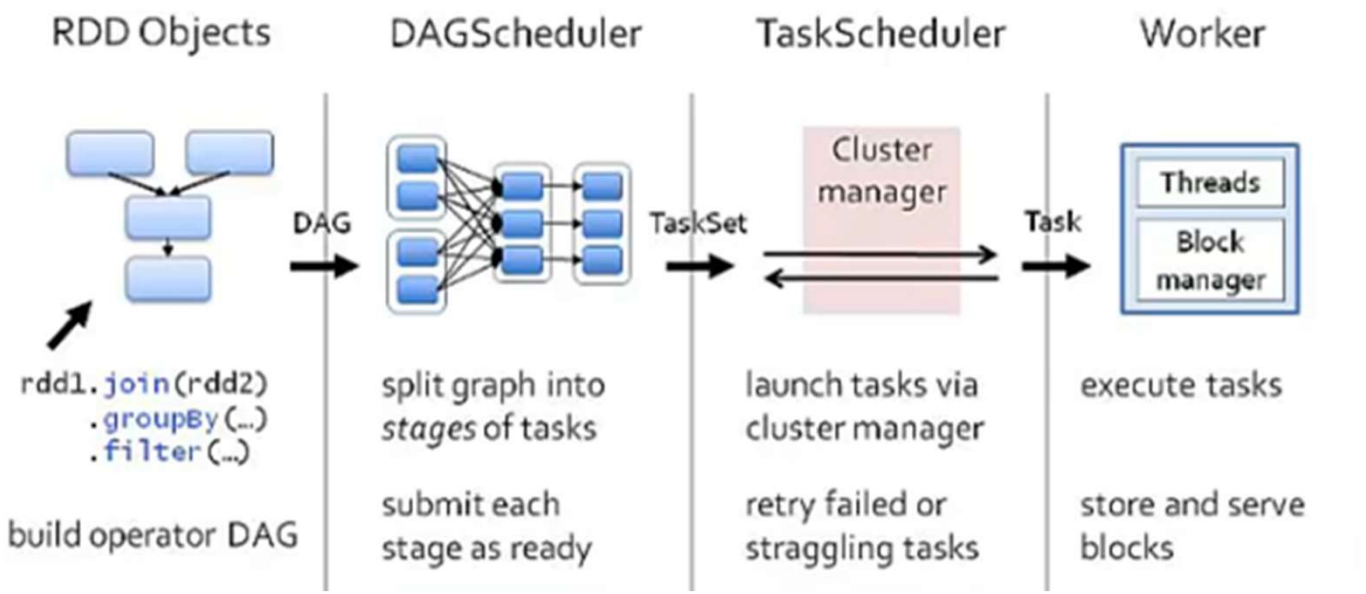
- Parent RDD/File storage
- Function to compute partition given parents
- Partitions
  - Metadata
  - Each partition is processed by a single task

# Outline

1. Breaking down computation tasks with Map, Shuffle, Reduce
2. Intuition behind RDDs
    a. Where the idea comes from
    b. How they are implemented
3. **Spark Scheduler**

# Scheduler

- How does Spark go from a collection of RDDs to a result?



# DAGScheduler

- Triggered by an action (ie collect() or count() method on an RDD)
- Takes in the DAG
- Determines stages by finding shuffle boundaries
  - reduceByKey, join, groupByKey
- Passes stages (sets of tasks) to the Task Scheduler
- Fault tolerance
  - Responsible for walking through the lineage if a worker goes down, starts recomputation

# DAGScheduler

User_id, location, activity_count
1,New York,5 2,London,0
3,Sydney,10 4,New York,3
5,London,8

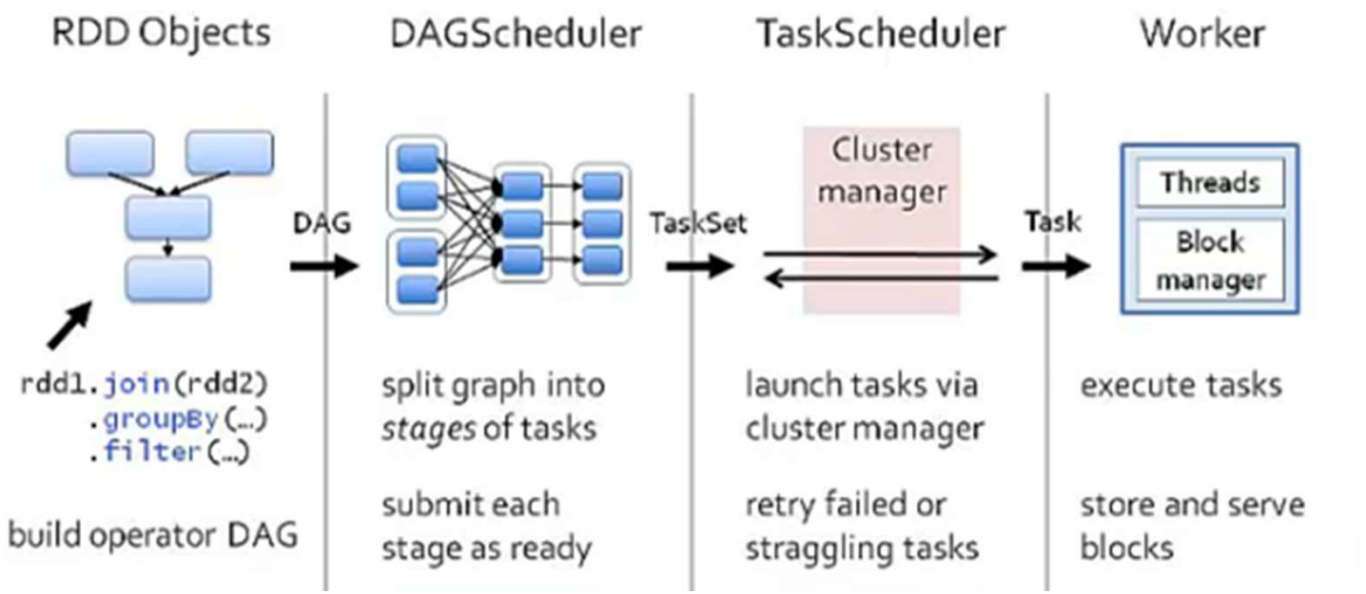
```
val logsRDD = sc.textFile("user_logs.csv")
val activeUsersRDD = logsRDD.filter(log => log.split(",")(2).toInt > 0)
val locationActivityRDD = activeUsersRDD.map(log => {
  val fields = log.split(",")
  (fields(1), fields(2).toInt) // (location, activity)
})
val totalActivityByLocationRDD = locationActivityRDD.reduceByKey(_ + _)
val sortedResultsRDD = totalActivityByLocationRDD.sortBy(_._2, ascending = false)

val result = sortedResultsRDD.collect()
```

Stage 1: Filter and Map
Stage 2: Reduce by key
Stage 3: Sort by key

# Scheduler

- How does Spark go from a collection of RDDs to a result?



# TaskScheduler

- Takes in the Stage
- Decides which tasks to run on which workers based on:
  - Data locality
  - Resource availability

# TaskScheduler

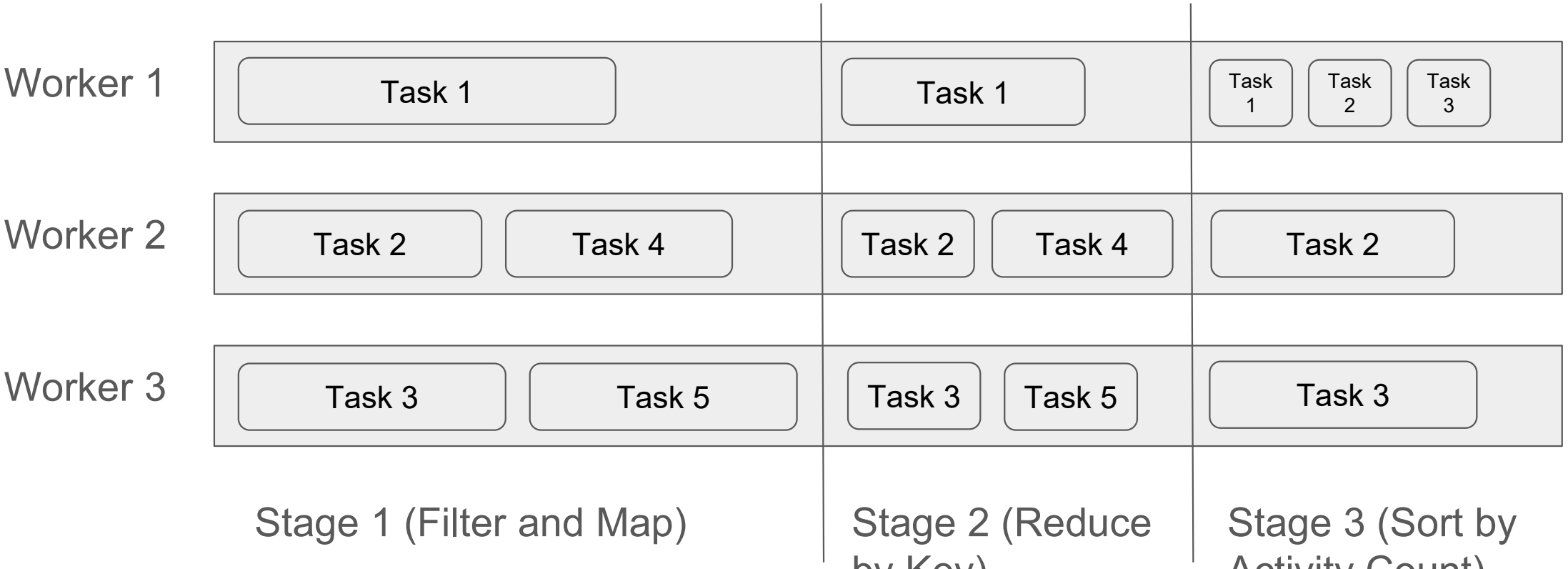
User_id, location, activity_count
1,New York,5 2,London,0
3,Sydney,10 4,New York,3
5,London,8

Stage 1: Filter and Map
Task 1: Read partition 1 of `logsRDD` , filter inactive users, map to `(location, activity)`
Task 2: Read partition 2 of `logsRDD` , filter inactive users, map to `(location, activity)`
And so on for all partitions of `logsRDD`

Stage 2: Reduce by key
Task 1: Reduce all records for locations assigned to partition 1.
Task 2: Reduce all records for locations assigned to partition 2.
And so on for all partitions.

Stage 3: Sort by key
Task 1: Sort records in partition 1 by total activity count
Task 2: Sort records in partition 2 by total activity count
And so on for all partitions

# Execution Overview




Assumes 5 partitions (task for each partition)

# Outline

1. Breaking down computation tasks with Map, Shuffle, Reduce
2. Intuition behind RDDs (recomputation for efficient fault tolerance)
3. Spark Scheduler