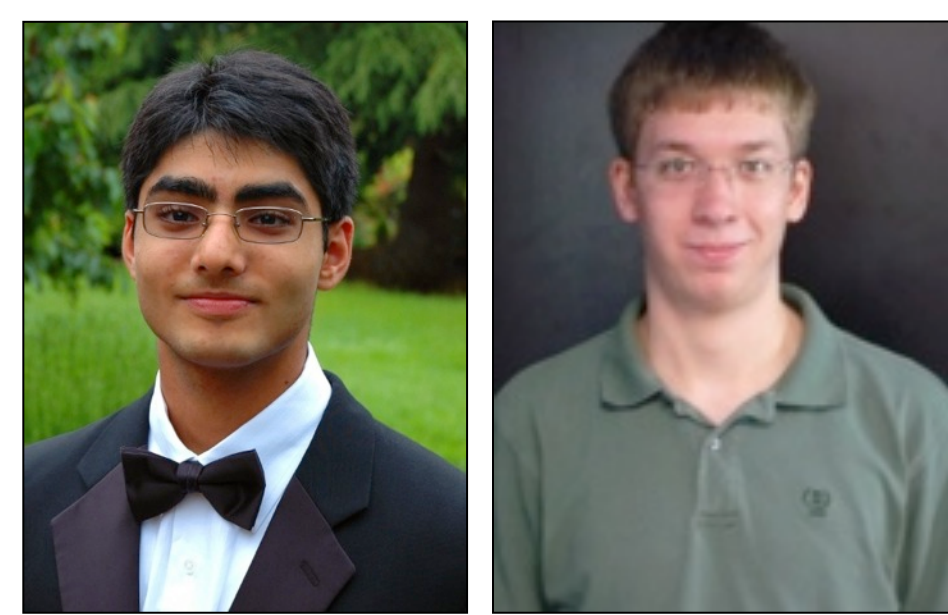


# Spark Debugger



Ankur Dave, Matei Zaharia, Murphy McCauley, Scott Shenker, Ion Stoica

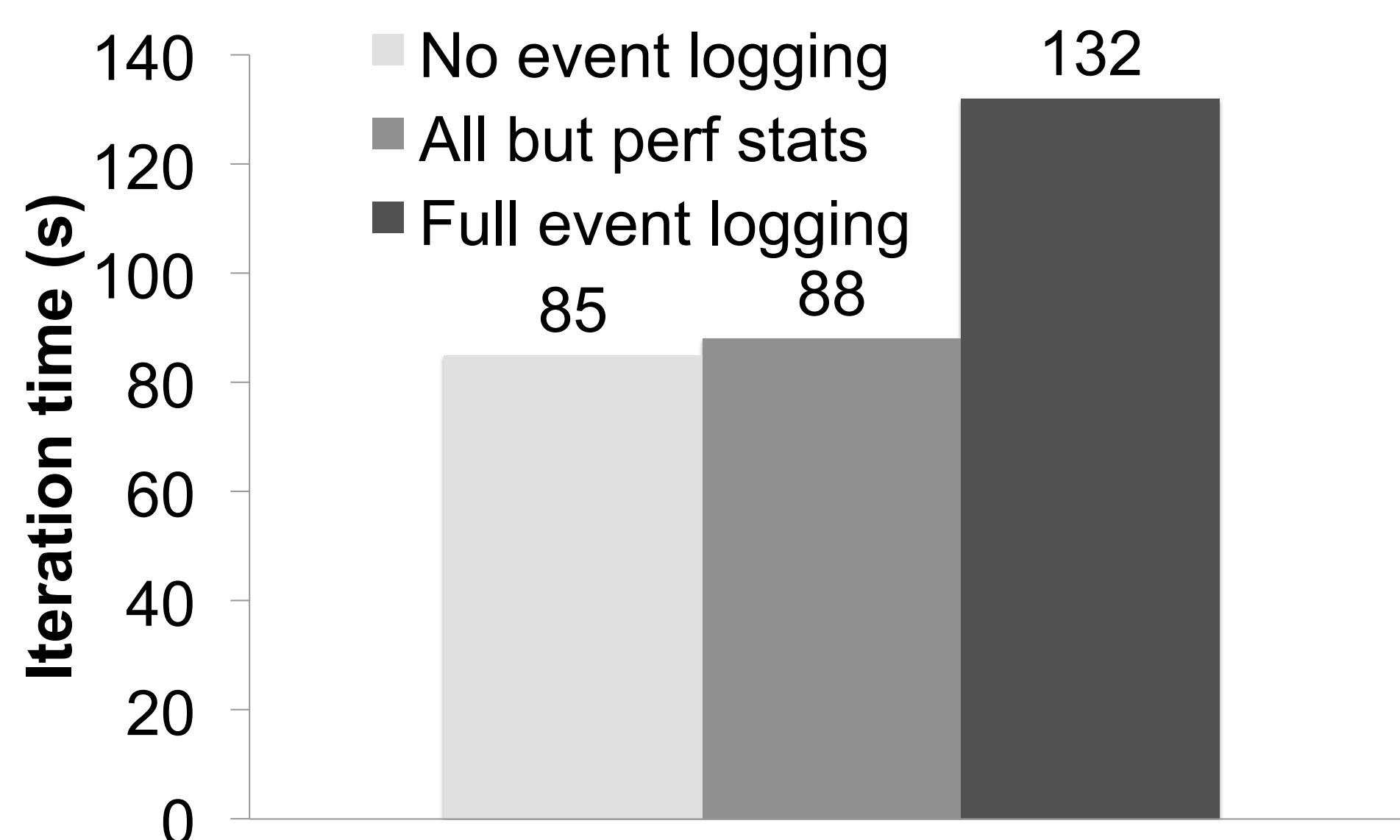
## Motivation

- People want to be able to monitor and debug Spark programs
- Conventional debugging incurs too much overhead on a cluster
- Can we use the structure of Spark programs to debug and replay with minimal overhead?

## Approach

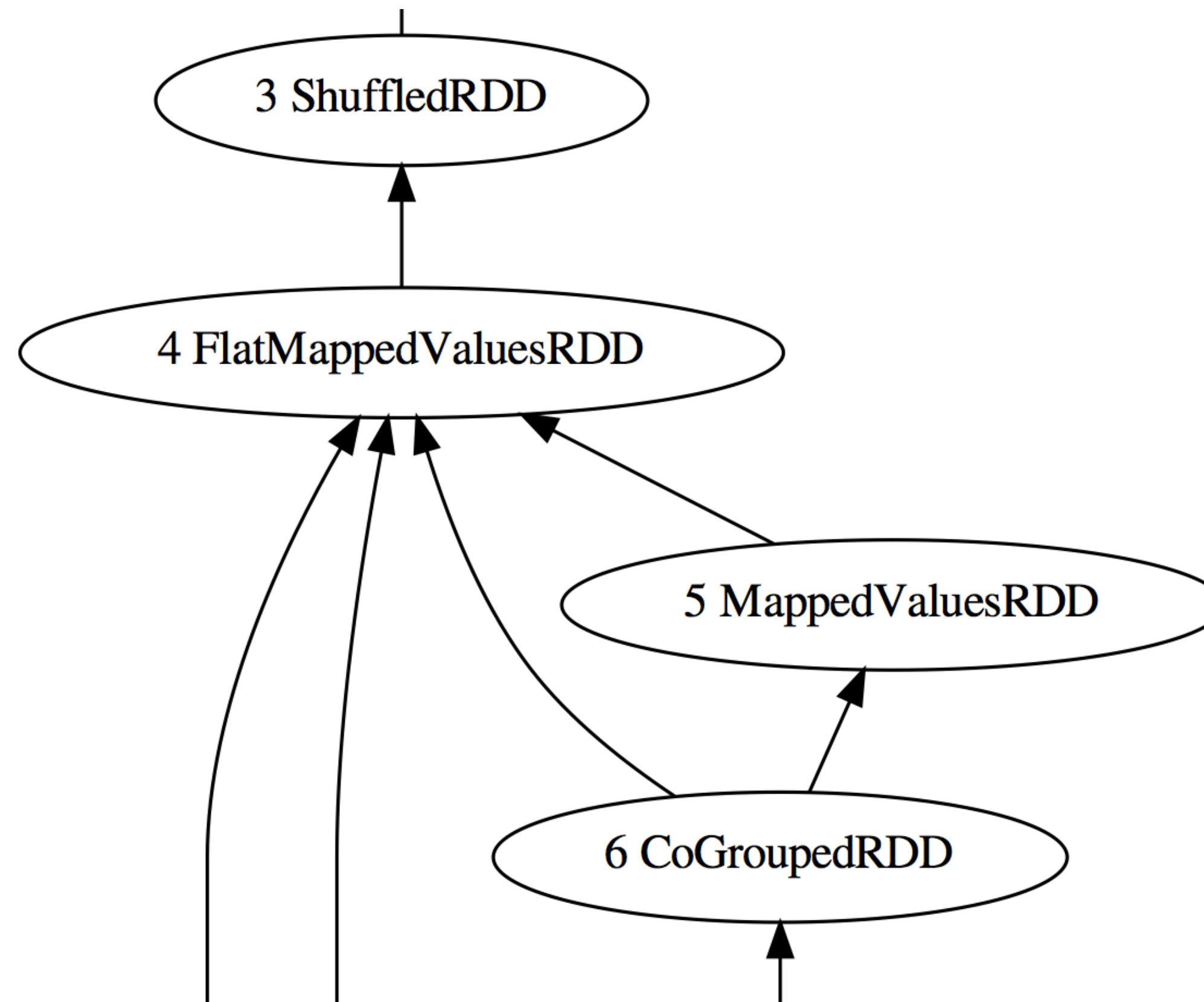
- As Spark program runs, workers report key events back to the master
- RDD lineage, checksums, shuffle order, and other events are logged
- Important events like errors are shown immediately to the user
- Later, user can reconstruct and query intermediate RDDs using the logged information, and run tasks in a conventional debugger

## Performance



## Capabilities

- *RDD replay and inspection*: interactively query and visualize intermediate RDDs



- *Exception monitoring*: aggregate and display exceptions thrown on slaves

```
01/09 03:54:02 INFO spark.SimpleJob: Starting task 0:19 as TID 19 on slave 201
01/09 03:54:06 INFO spark.SimpleJob: Lost TID 0 (task 0:0)
01/09 03:54:06 INFO spark.SimpleJob: Loss was due to java.lang.ArithmeticExcep
at spark.examples.ExceptionHandlingTest$$anonfun$1.apply$mcII$sp(Exceptio
at spark.examples.ExceptionHandlingTest$$anonfun$1.apply(ExceptioHandlir
at spark.examples.ExceptionHandlingTest$$anonfun$1.apply(ExceptioHandlir
at scala.collection.Iterator$$anon$19.next(Iterator.scala:335)
at ...
```

- *Performance monitoring*: find slow tasks caused by workload imbalance

```
scala> {
  println("RDD\tNormalized variance");
  println("-----\t-----");
  r.events.collect { case rs: RuntimeStatistics if rs.mean != 0 => rs }
    .groupBy(_.rddId)
    .mapValues(xs => stdDev(xs.map(_.mean)) / mean(xs.map(_.mean)))
    .toList.sortBy(_._2).reverse
    .foreach(x => println("#" + x._1 + "\t" + x._2))
}

RDD      Normalized variance
----
#4       4.611228850686498
#7       2.005886866272438
#11      1.708379166955671
#0       1.0156906223457116
#22      0.7463381659054615
#12      0.7310855018089487
#10      0.6380130541752221
```

- *Step-through debugging*: recompute a slice of an RDD in a conventional debugger

```
12/01/09 05:52:46 INFO spark.BoundedMem [root@ip-10-114-54-208 ~]# jdb -attach 8000
12/01/09 05:52:46 INFO spark.CacheTrack Set uncaught java.lang.Throwable
12/01/09 05:52:46 INFO spark.MapOutputT Set deferred uncaught java.lang.Throwable
12/01/09 05:52:47 INFO spark.SparkEnv: Initializing jdb ...
[GENERIC] [1/9/12 5:52 AM] [RemoteServe >
12/01/09 05:52:47 INFO spark.EventReport VM Started: No frames on the current call sta
Spark context available as sc.
Type in expressions to have them evalua main[1] use /root/spark/core/src/main/scala:/
Type :help for more information. stop at spark.examples.ExceptionHandlingTest$
scala> val r = new spark.EventLogReader main[1] Deferring breakpoint spark.examples.E
r: spark.EventLogReader = spark.EventLo It will be set after the class is loaded.
main[1] cont
> Set deferred breakpoint spark.examples.Exce
scala> val ex = r.events.collectFirst { Breakpoint hit: "thread=Thread-27", spark.exa
ex: spark.ExceptionEvent = ExceptionEve 13 val result = sc.parallelize(0 to 50
scala> r.debugException(ex) Thread-27[1]
Running task ResultTask(0, 0)
0 bash 1 bash 2 bash 3 bash 4 bash 5 bash 6 bash 7 bash 8 bash 9 bash 10 bash
```

- *Serialization overhead monitoring*: calculate how much time is lost to serialization
- *Determinism verification*: ensure that user code is deterministic

## Future Plans

- *Culprit determination*: automatically find which element of an RDD is causing a crash
- *GC monitoring*: determine whether GC is a problem and what type of objects are being collected
- *Cache monitoring*: determine what is being inserted and evicted from each machine's cache