Titian: Data Provenance Support in Spark

Tom Meagher

&

Fangling Zhang

What’s Titian

- A library that debug data quickly in Apache Spark
- Enables *data provenance* by obtaining a LineageRDD reference from any given RDD

Why Titian

The limitations of current approach (RAMP and Newt) of supporting data lineage in DISC systems (Hadoop or Spark):

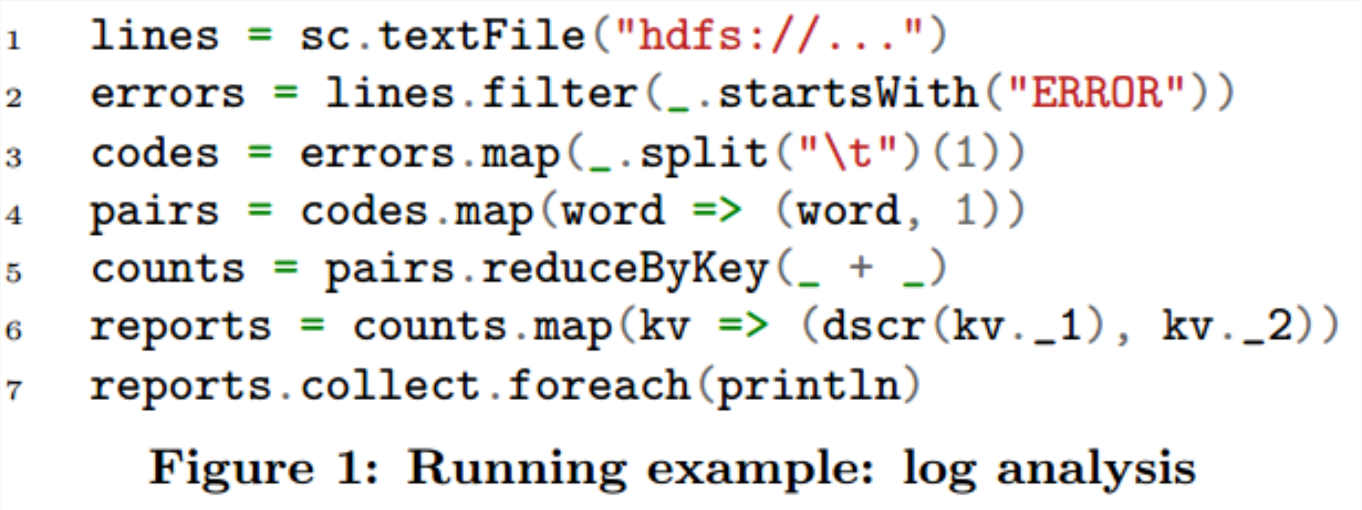
- Use external storage (sharded DBMS or HDFS) to retain lineage information
- supported in a separate programming interface
- Little support for viewing or replaying intermediate data

Titian’s Contributions

- A data lineage capture and query support system in Apache Spark.
- Exhibit an overhead of less than 30%.
- Interactive data provenance query support that extends the familiar Spark RDD programming model.
- A variety of design alternatives for capturing and tracing data lineage

Running example: log analysis

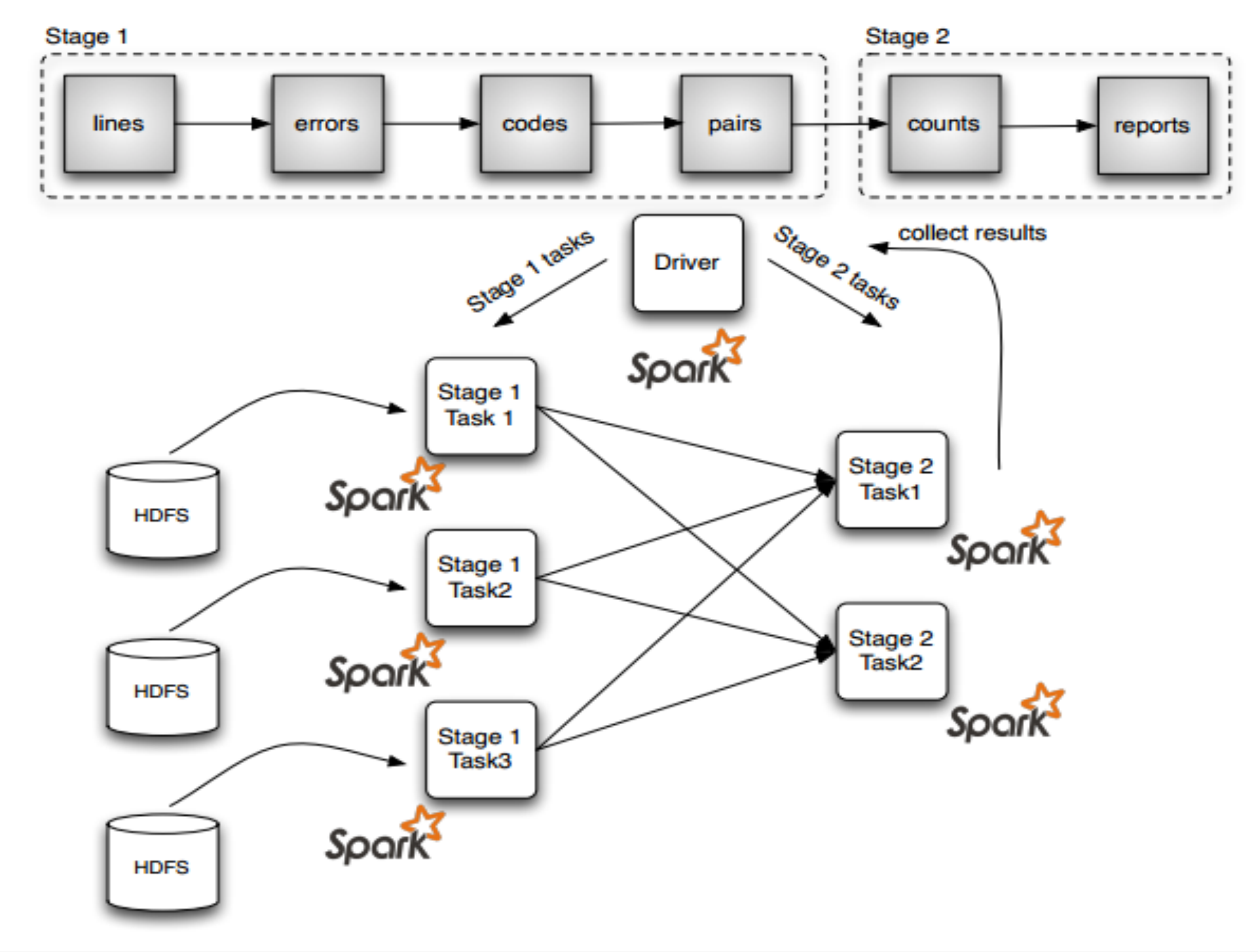
- Obtain a report containing the description of each error, together with its count.



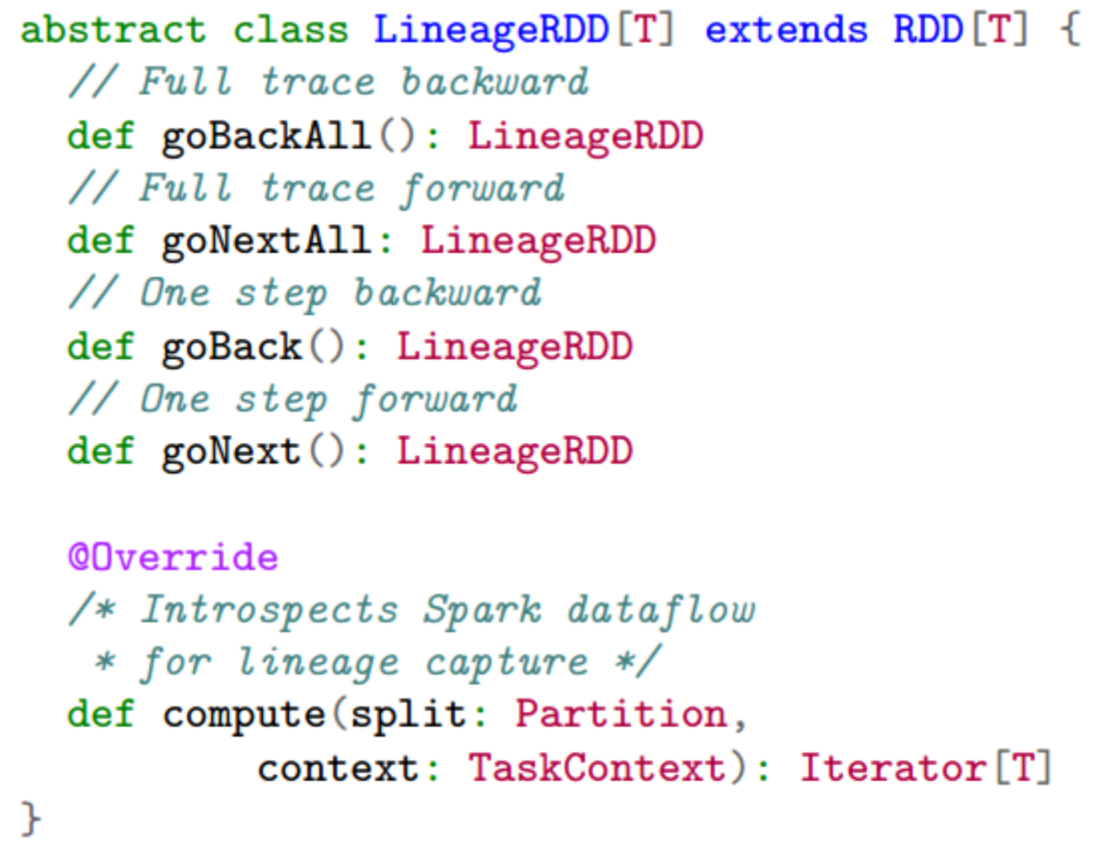
```
lines = sc.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
codes = errors.map(_.split("\t")(1))
pairs = codes.map(word => (word, 1))
counts = pairs.reduceByKey(_ + _)
reports = counts.map(kv => (dscr(kv._1), kv._2))
reports.collect.foreach(println)
```

Figure 1: Running example: log analysis

Running example: log analysis



Titian: Details



```
abstract class LineageRDD[T] extends RDD[T] {
  // Full trace backward
  def goBackAll(): LineageRDD
  // Full trace forward
  def goNextAll: LineageRDD
  // One step backward
  def goBack(): LineageRDD
  // One step forward
  def goNext(): LineageRDD

  @Override
  /* Introspects Spark dataflow
   * for lineage capture */
  def compute(split: Partition,
           context: TaskContext): Iterator[T]
}
```

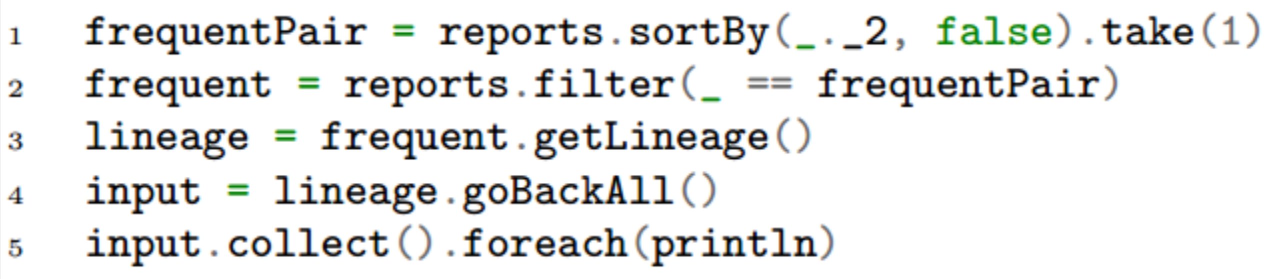
- **LineageRDD** application programming interface, which extends the RDD abstraction with tracing capabilities.

- **goBackAll, goNextAll**: to compute the full trace backward and forward respectively

- **goBack and goNext** :A single step backward or forward

Titian Application Examples

- **Example 1: Backward Tracing** -selects the most frequent error, then traces back to the input lines containing such errors and prints them.

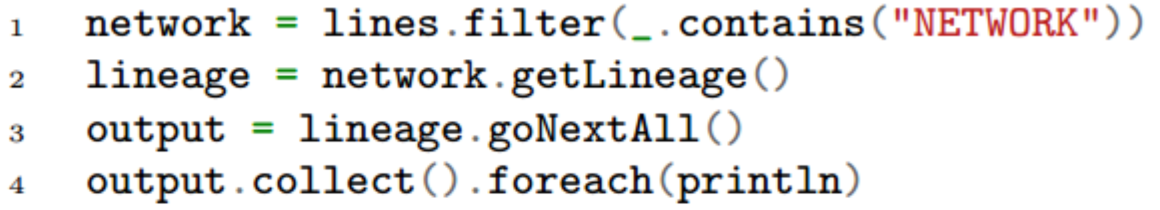


```
frequentPair = reports.sortBy(_._2, false).take(1)
frequent = reports.filter(_ == frequentPair)
lineage = frequent.getLineage()
input = lineage.goBackAll()
input.collect().foreach(println)
```

Figure 5: Input lines with the most frequent error

Titian Application Examples

- **Example 2: Forward Tracing** - find the error codes generated from the network sub-system.

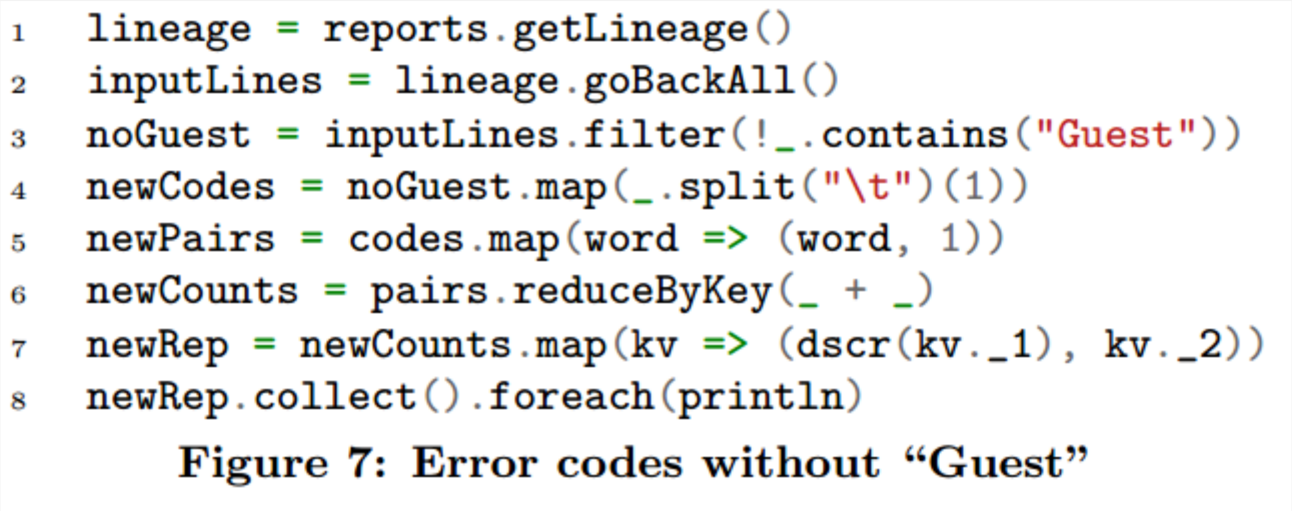


```
network = lines.filter(_.contains("NETWORK"))
lineage = network.getLineage()
output = lineage.goNextAll()
output.collect().foreach(println)
```

Figure 6: Network-related error codes

Titian Application Examples

- **Example 3: Selective Replay** - seeing the errors distribution without the ones caused by “Guest."



```
lineage = reports.getLineage()
inputLines = lineage.goBackAll()
noGuest = inputLines.filter(!_.contains("Guest"))
newCodes = noGuest.map(_.split("\t")(1))
newPairs = codes.map(word => (word, 1))
newCounts = pairs.reduceByKey(_ + _)
newRep = newCounts.map(kv => (dscr(kv._1), kv._2))
newRep.collect().foreach(println)
```

Figure 7: Error codes without “Guest”

Titian’s Capture Points

Titian captures data lineage (from Spark’s stage DAG) in three places:

- **Input:** Data imported from some external source *e.g.*, HDFS, Java Collection, etc.
- **Stage:** The output of a stage executed by a task.
- **Aggregate:** In an aggregation operation *i.e.*, combiner, group-by, reduce, and join.

Titian’s agents

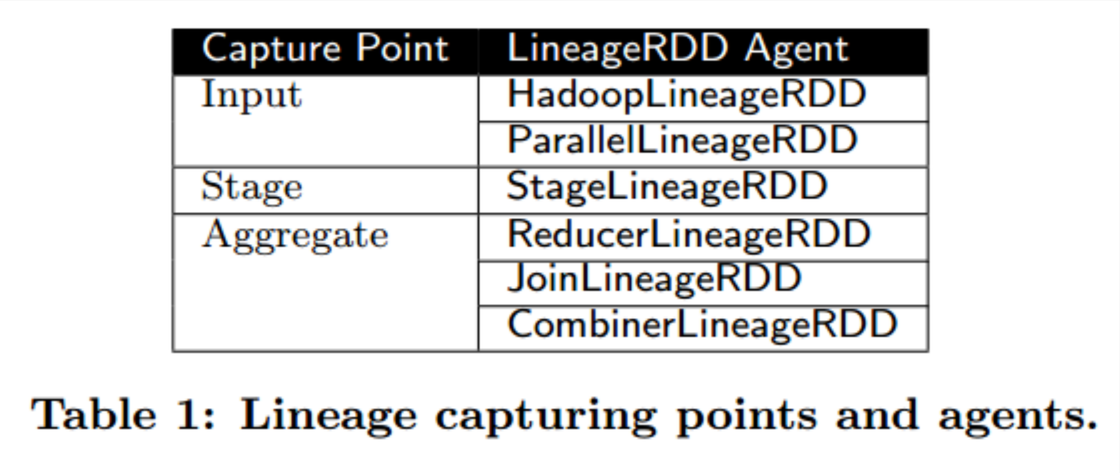
Titian uses agents to capture data lineage. The responsibility of these agents:

- To generate unique identifiers for each new record
- Associate output records of a given operation (*i.e.*, stage, shuffle step) with relevant input records

Titian’s agents

Titian uses agents to capture data lineage. The responsibility of these agents:

- To generate unique identifiers for each new record
- Associate output records of a given operation (*i.e.*, stage, shuffle step)with relevant input records

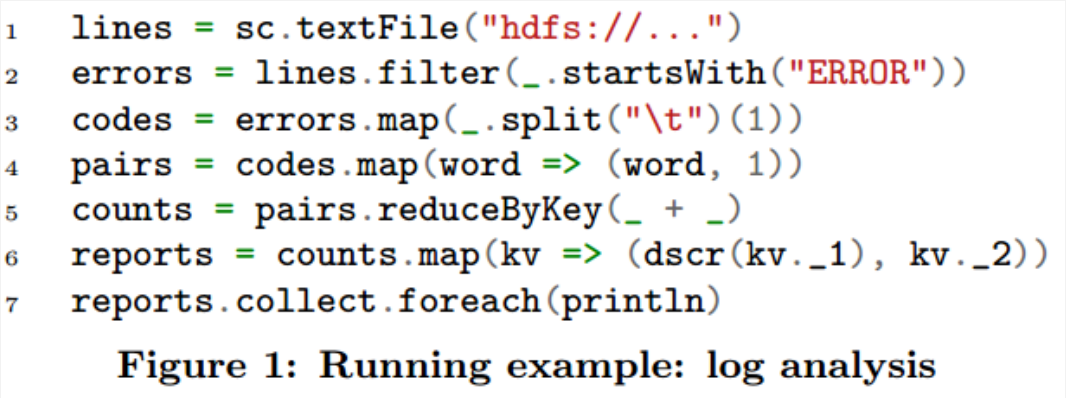


Capture Point	LineageRDD Agent
Input	HadoopLineageRDD
	ParallelLineageRDD
Stage	StageLineageRDD
Aggregate	ReducerLineageRDD
	JoinLineageRDD
	CombinerLineageRDD

Table 1: Lineage capturing points and agents.

Dataflow Instrumentation

Example 4: Returning to our running example…



```
lines = sc.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
codes = errors.map(_.split("\t")(1))
pairs = codes.map(word => (word, 1))
counts = pairs.reduceByKey(_ + _)
reports = counts.map(kv => (dscr(kv._1), kv._2))
reports.collect.foreach(println)
```

Figure 1: Running example: log analysis

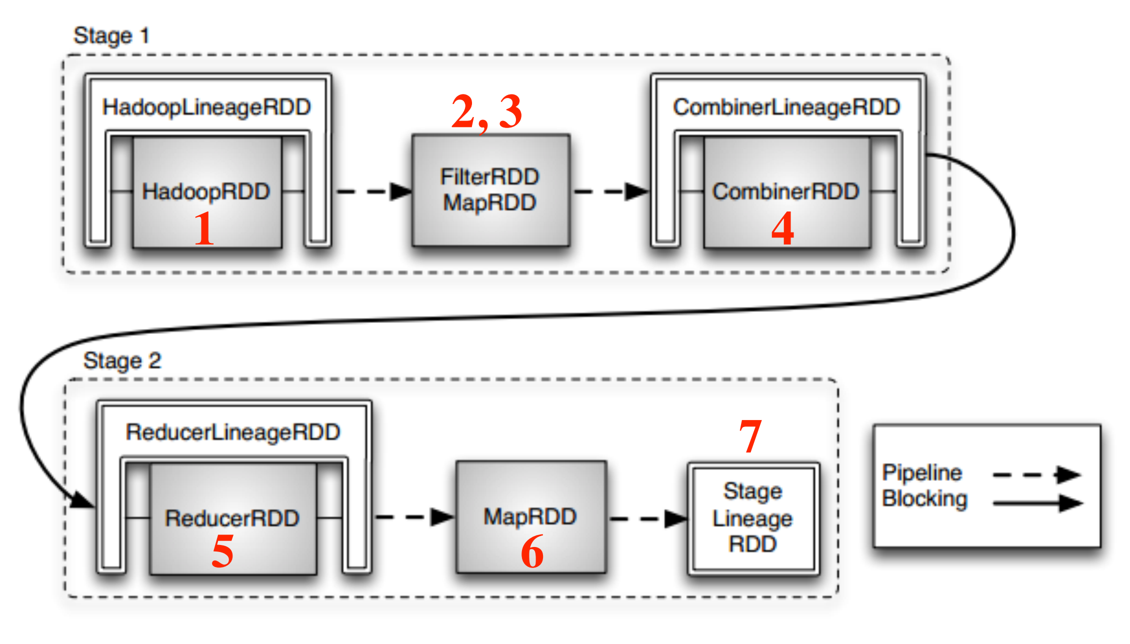


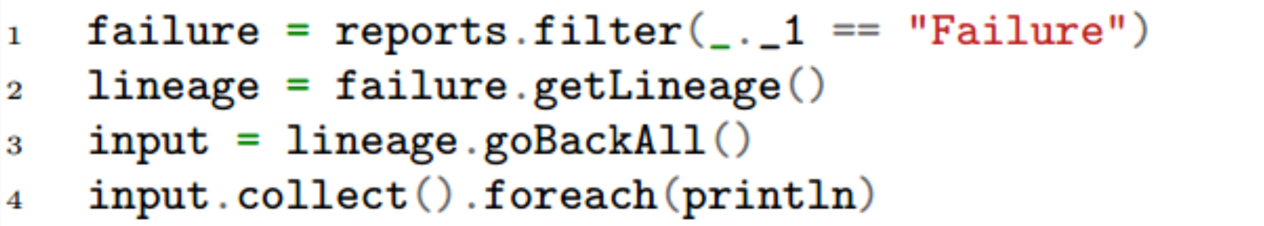
Figure 8: Job workflow after adding the lineage capture points

Lineage Storage

- Titian stores all data lineage in the **BlockManager** (Spark’s internal storage layer for intermediate data).
- The agents’ associations are stored in a BlockManager table, which defines two columns containing the:
    - (1) input record identifiers, and
    - (2) output record identifiers. (*Tracing occurs by recursively joining the tables)

Querying the Lineage Data

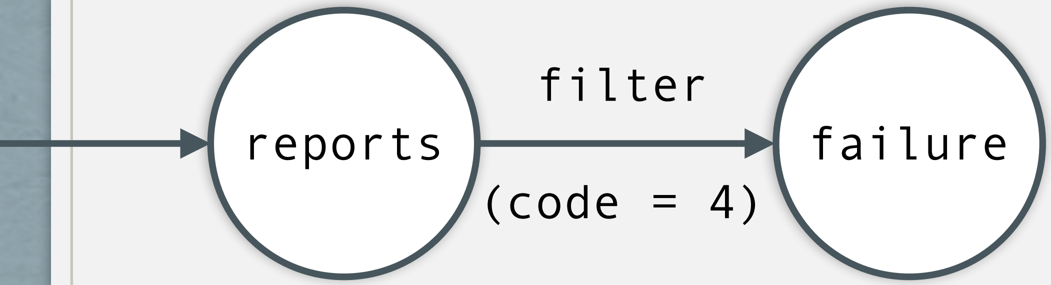
- **Example 5:** To trace back and see the actual log entries that correspond to a “Failure" (code =4)



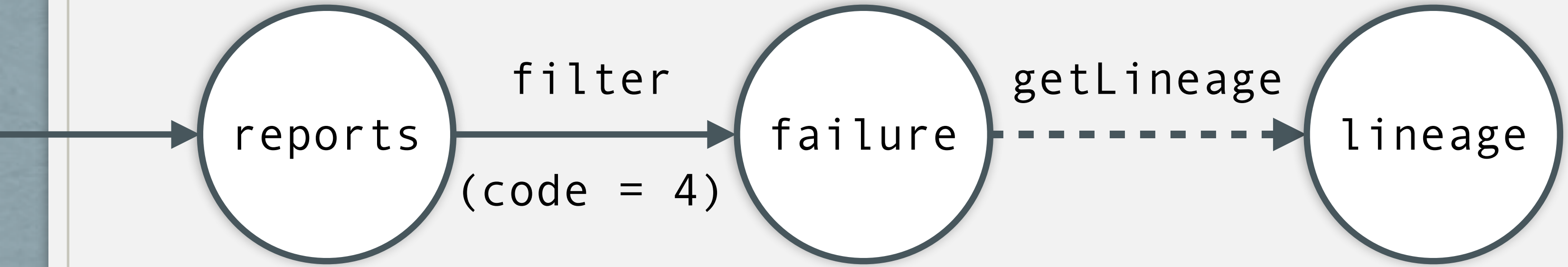
```
failure = reports.filter(_._1 == "Failure")
lineage = failure.getLineage()
input = lineage.goBackAll()
input.collect().foreach(println)
```

Figure 10: Tracing backwards the “Failure” errors

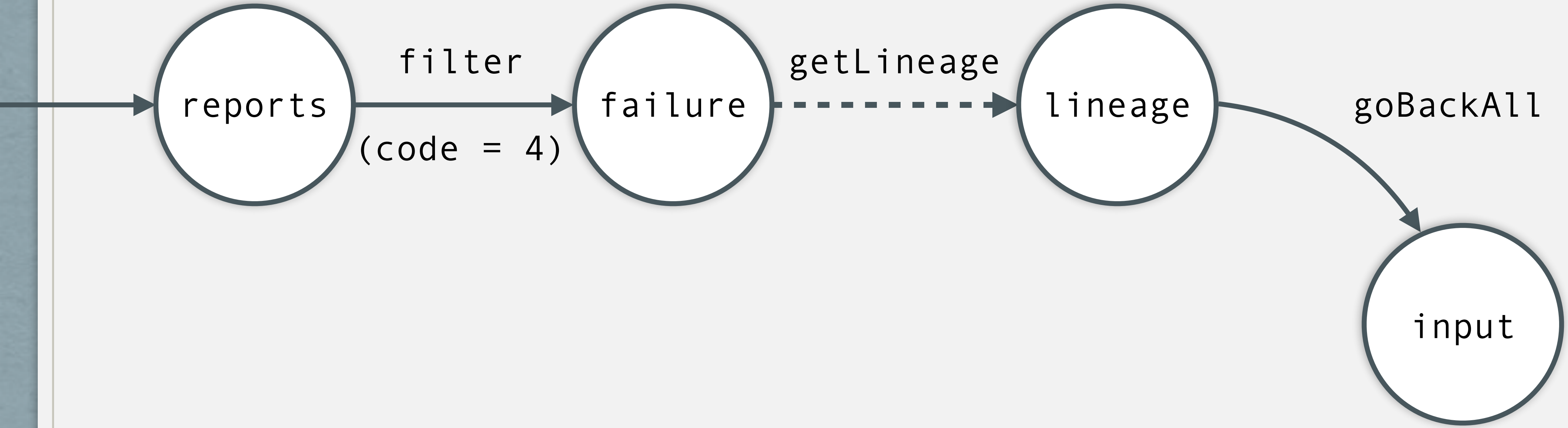
```
failure = reports.filter(_._1 == “Failure”)
```



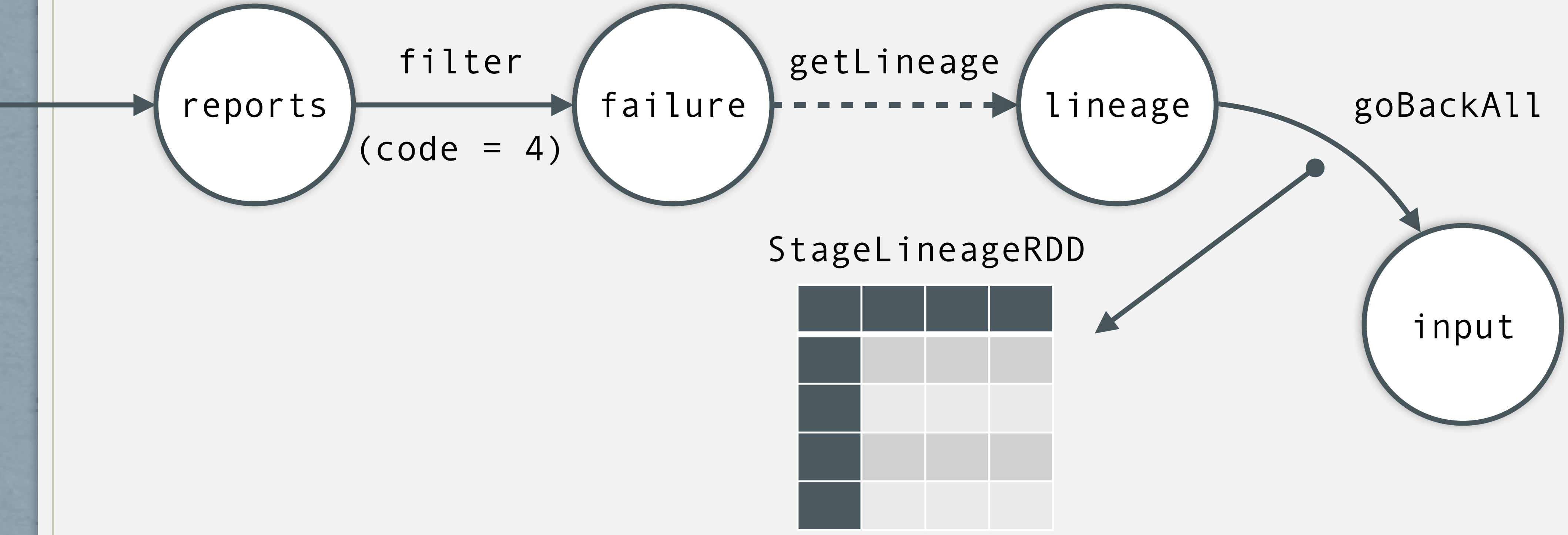
lineage = failure.getLineage()



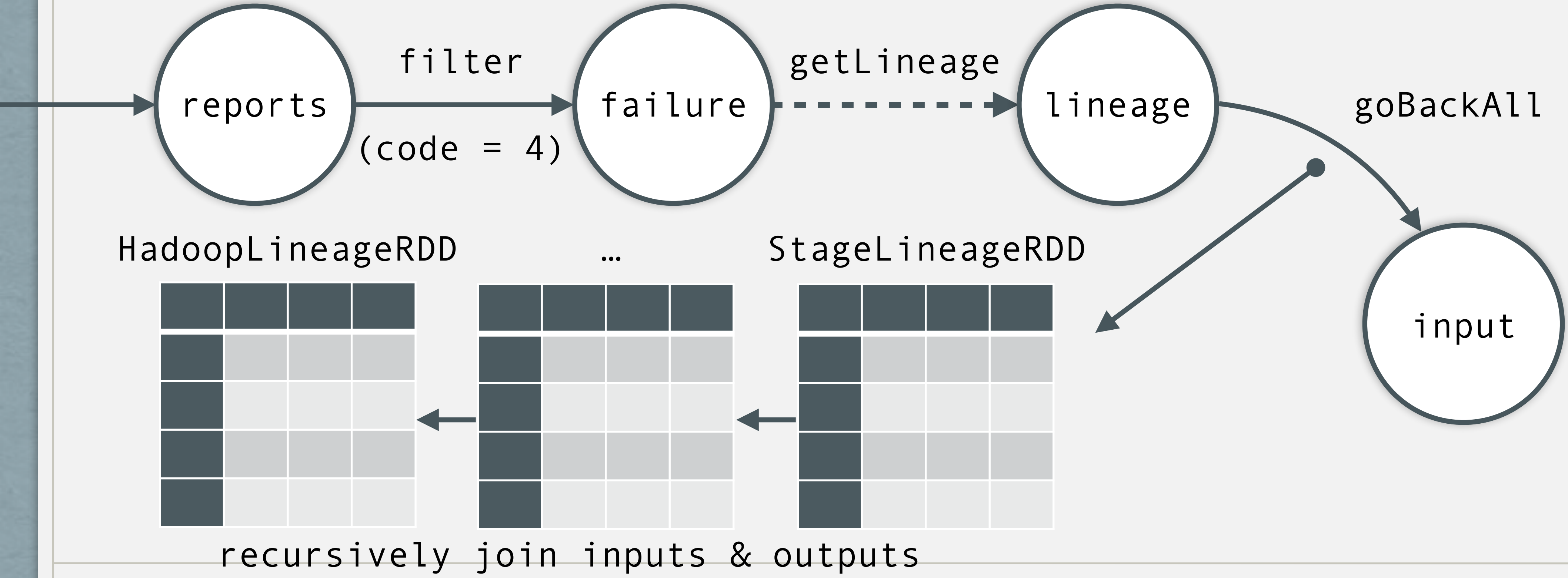
```
input = lineage.goBackAll()
```



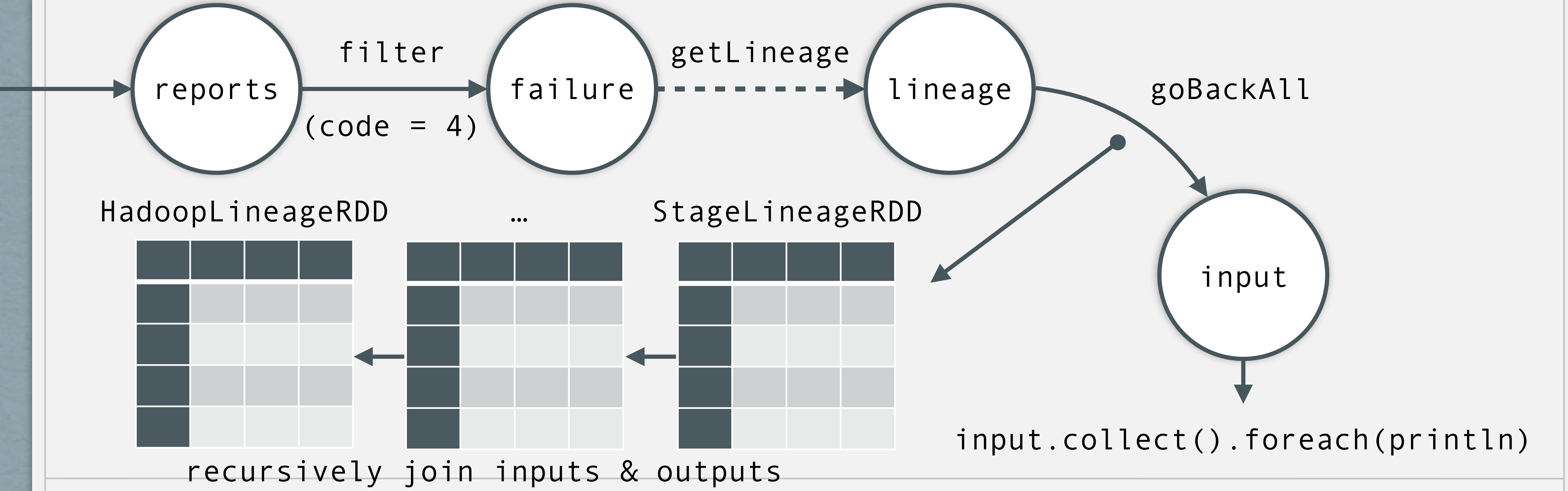
input = lineage.goBackAll()



input = lineage.goBackAll()

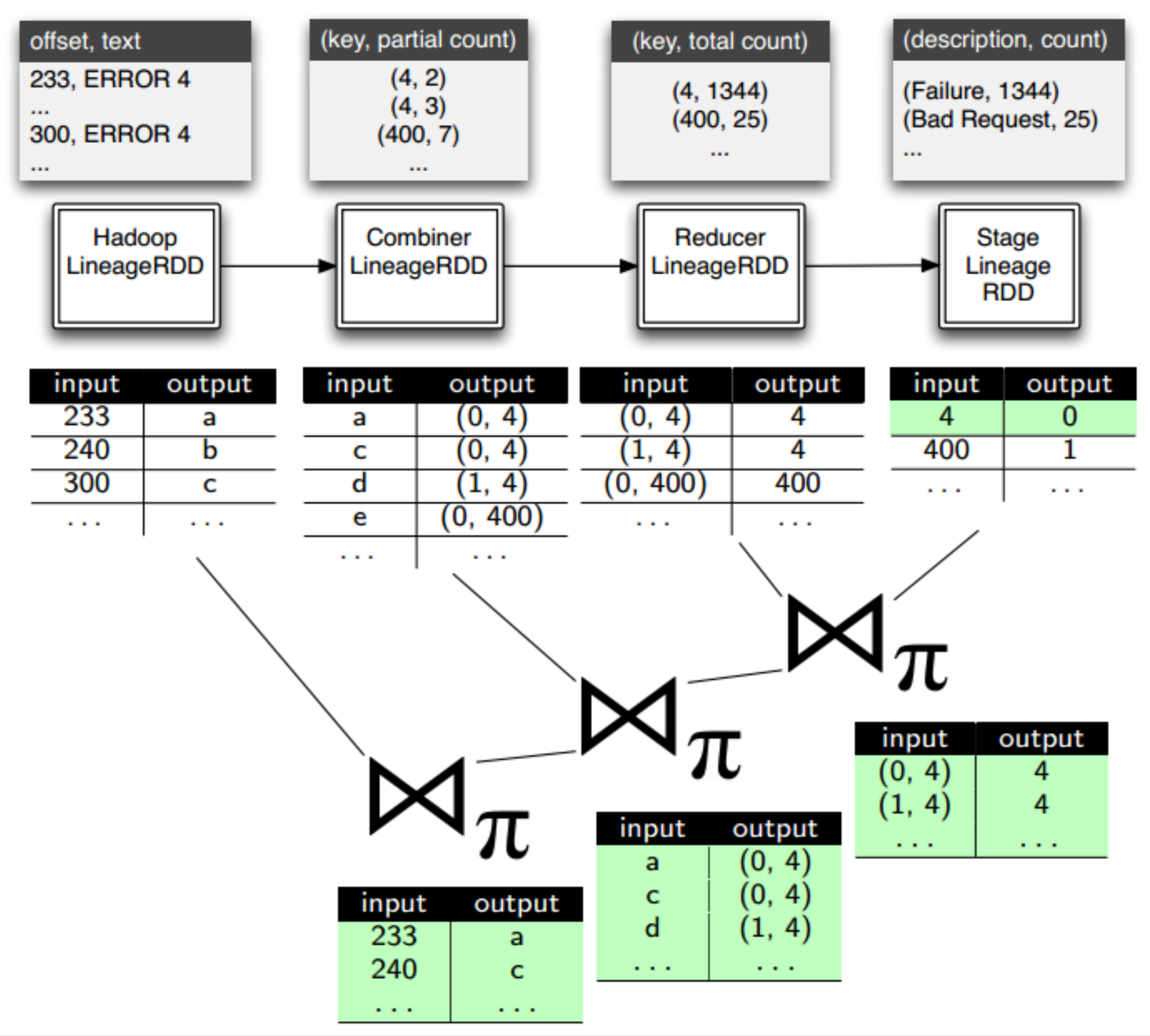


input = lineage.goBackAll()



Querying the Lineage Data

- **Example 5: A logical trace plan** that recursively joins data lineage tables



offset, text
233, ERROR 4
...
300, ERROR 4
...

(key, partial count)
(4, 2)
(4, 3)
(400, 7)
...

(key, total count)
(4, 1344)
(400, 25)
...

(description, count)
(Failure, 1344)
(Bad Request, 25)
...

input	output
233	a
240	b
300	c
. . .	. . .

input	output
a	(0, 4)
c	(0, 4)
d	(1, 4)
e	(0, 400)
. . .	. . .

input	output
(0, 4)	4
(1, 4)	4
(0, 400)	400
. . .	. . .

input	output
4	0
400	1
. . .	. . .

input	output
(0, 4)	4
(1, 4)	4
. . .	. . .

input	output
a	(0, 4)
c	(0, 4)
d	(1, 4)
. . .	. . .

input	output
233	a
240	c
. . .	. . .

Titian’s Fault-tolerance

- Titian does not break the fault tolerance model of Spark.
- During the tracing phase, LineageRDDs behave as a normal RDD and, as such, are resilient to failures.

Titian’s Alternative Designs

- Titian-D: stores data lineage *distributed* in the BlockManager local to the capture agent.

Titian’s Alternative Designs

- Titian-D: stores data lineage *distributed* in the BlockManager local to the capture agent.

Titian can easily implement other debugging strategies:

- **Titian-P**: each capturing agent generates the lineage data and without storing it; lineage references are instead appended to a list and *propagated* downstream. The final stage capture point will then store the complete lineage data.

Titian’s Alternative Designs

- Titian-D: stores data lineage *distributed* in the BlockManager local to the capture agent.

Titian can easily implement other debugging strategies:

- Titian-P: each capturing agent generates the lineage data and without storing it; lineage references are instead appended to a list and *propagated* downstream. The final stage capture point will then store the complete lineage data.

- **Titian-C**: saves all the lineage into a unique *centralized* server in its local file system.

Titian’s Alternative Designs

- Titian-D: stores data lineage *distributed* in the BlockManager local to the capture agent.

Titian can easily implement other debugging strategies:

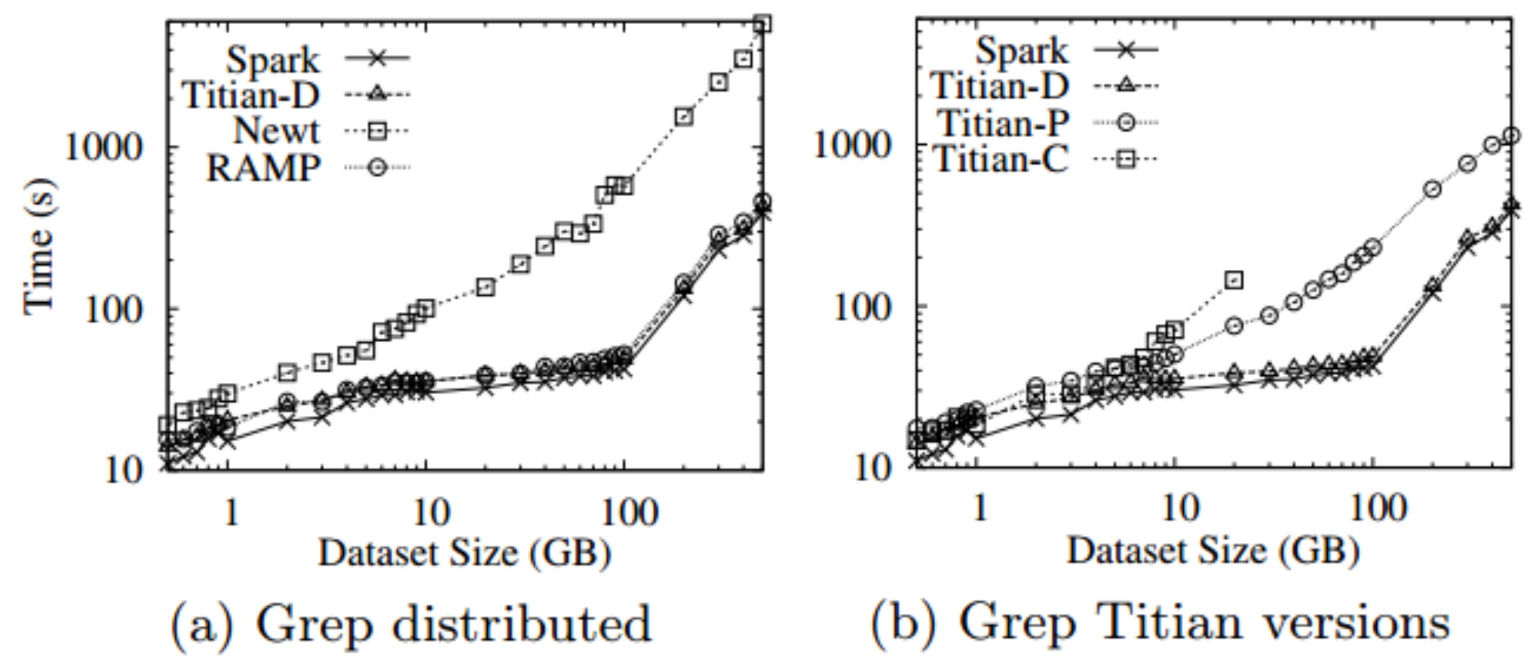
- Titian-P: each capturing agent generates the lineage data and without storing it; lineage references are instead appended to a list and *propagated* downstream. The final stage capture point will then store the complete lineage data.

- Titian-C : saves all the lineage into a unique *centralized* server in its local file system.

- Both Titian-C and Titian-P tradeoff space overheads, by aggregating lineage data into a more centralized storage, for a faster tracing time.

Experimental Evaluation

- Data Lineage Capture Overheads Comparison

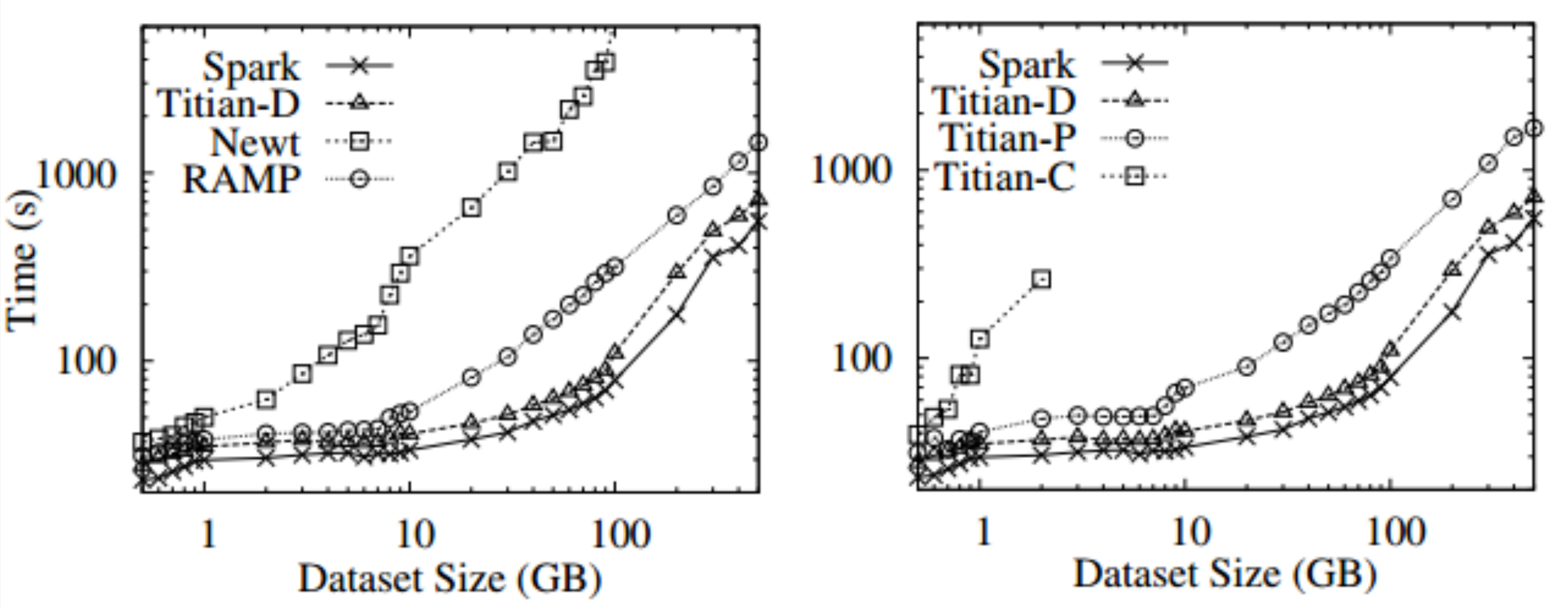


(a) Grep distributed

(b) Grep Titian versions

Experimental Evaluation

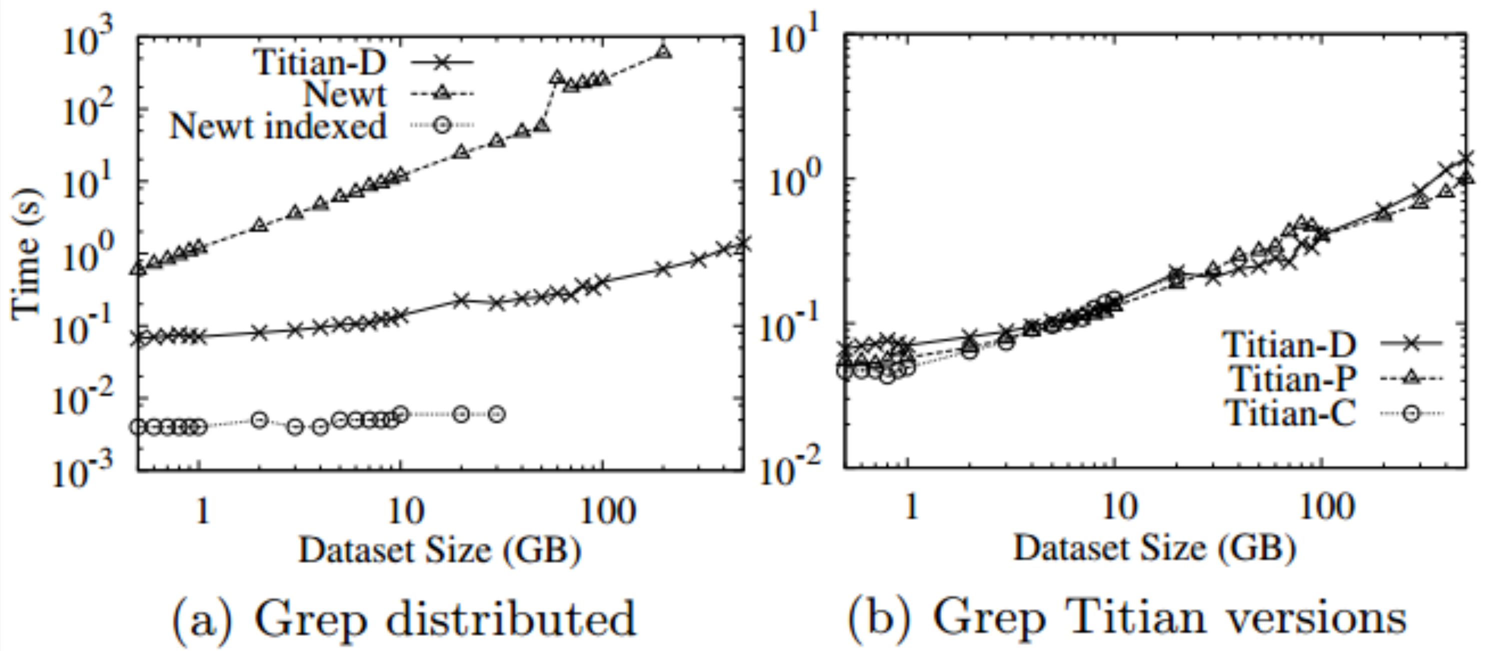
- Data Lineage Capture Overheads Comparison



(c) WordCount distributed (d) WordCount Titian versions

Experimental Evaluation

- Tracing Time Comparison

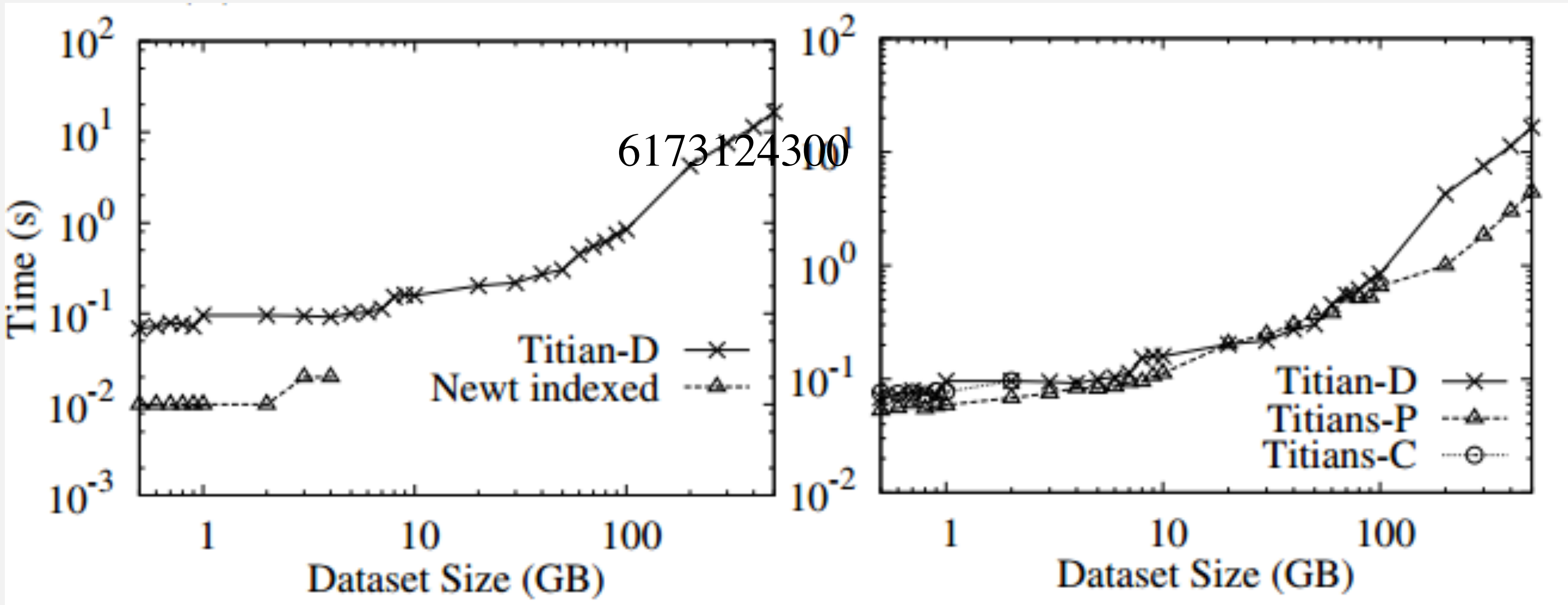


(a) Grep distributed

(b) Grep Titian versions

Experimental Evaluation

- Tracing Time Comparison



(c) WordCount distributed (d) WordCount Titian versions

Questions

Tom Meagher

&

Fangling Zhang