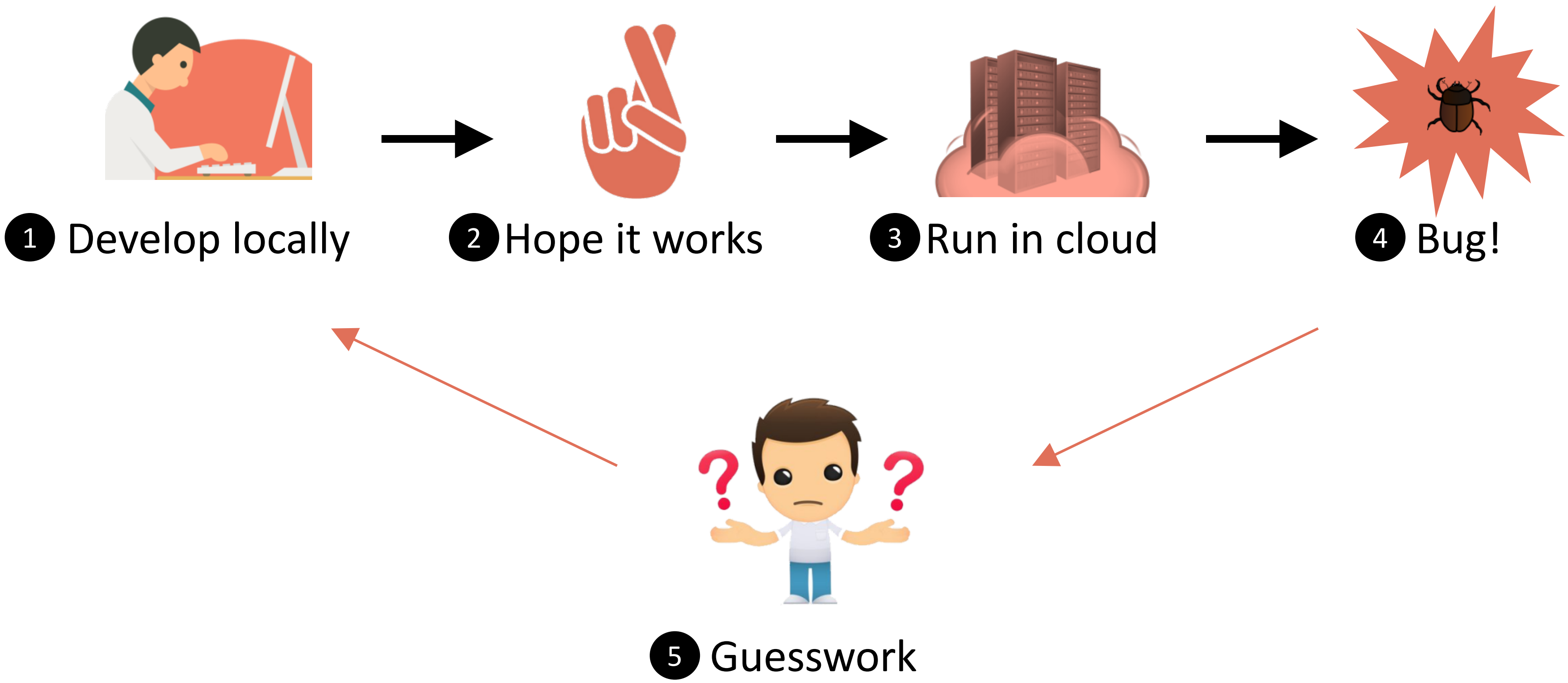
Automated Debugging In Data Intensive Scalable Computing Systems

Muhammad Ali Gulzar[1], Matteo Interlandi[3], Xueyuan Han[2], Mingda Li[1],
Tyson Condie[1], and Miryung Kim[1]

[1]University of California, Los Angeles
[2]Harvard University
[3]Mircrosoft

Big Data Debugging in the Dark



Map Reduce

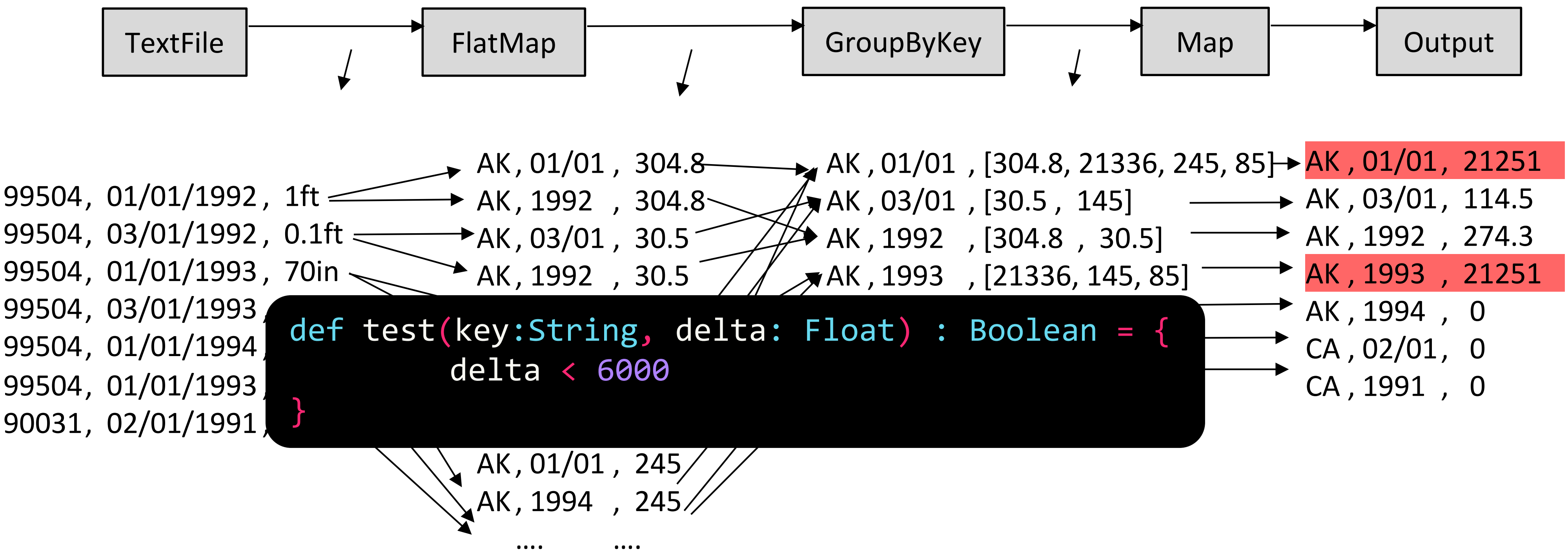
Motivating Example

- Alice writes a Spark program that identifies, **for each state** in the US, the **delta between the minimum and the maximum** snowfall reading for **each day of any year** and **for any particular year**.

Zip Code	Date	Snow Fall
99504	01/01/1994	245mm
99504	01/01/1993	85mm
90031	02/01/1991	0mm
…	…	…

Problem Definition

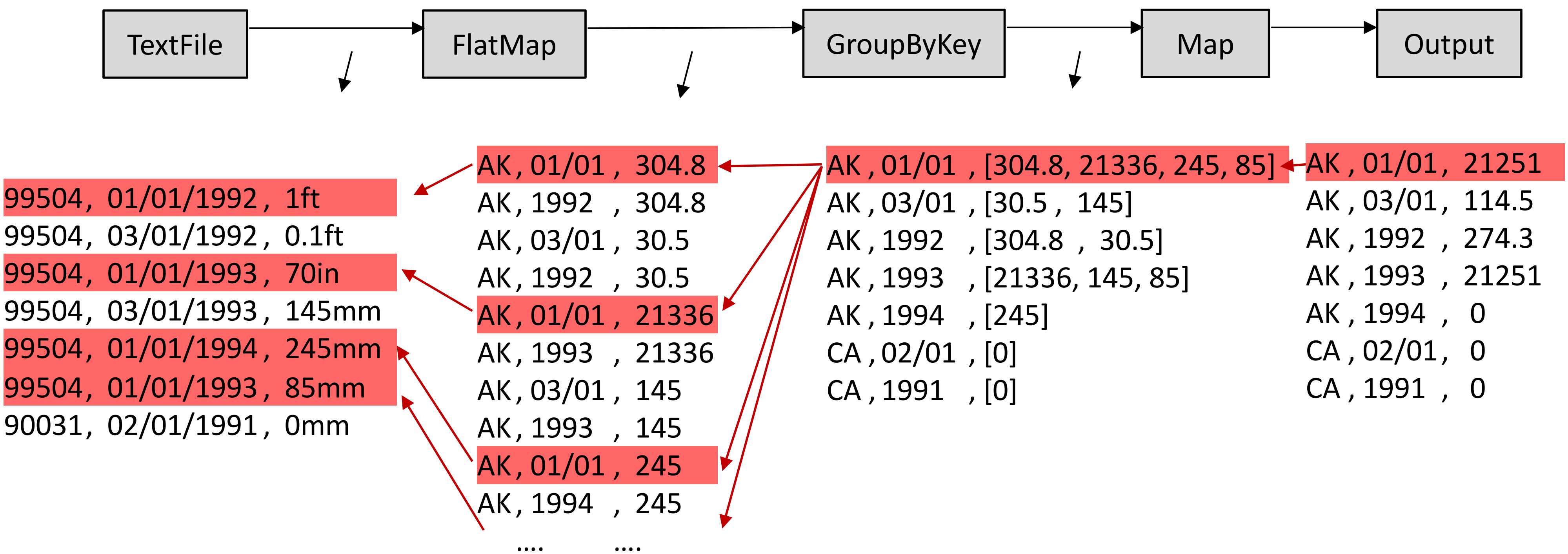
- Using a test function, a user can specify incorrect results




```
def test(key:String, delta: Float) : Boolean = {
        delta < 6000
}
```

Given a test function, the goal is to identify a minimum subset of the input that is able to reproduce the same test failure.

Existing Approach 1: Data Provenance for Spark



It over-approximates the scope of failure-inducing inputs *i.e.* records in the faulty key-group are all marked as faulty

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

1
99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
99504, 01/01/1993, 70in

2
99504, 03/01/1993, 145mm
99504, 01/01/1994, 245mm
99504, 01/01/1993, 85mm
90031, 02/01/1991, 0mm

AK , 01/01 , 304.8
AK , 1992 , 304.8
AK , 03/01 , 30.5
AK , 1992 , 30.5
AK , 01/01 , 21336
AK , 1993 , 21336
AK , 03/01 , 145
AK , 1993 , 145
AK , 01/01 , 245
AK , 1994 , 245
....

AK , 01/01 , [304.8, 21336, 245, 85]
AK , 03/01 , [30.5 , 145]
AK , 1992 , [304.8 , 30.5]
AK , 1993 , [21336, 145, 85]
AK , 1994 , [245]
CA , 02/01 , [0]
CA , 1991 , [0]

AK , 01/01 , 21251
AK , 03/01 , 114.5
AK , 1992 , 274.3
AK , 1993 , 21251
AK , 1994 , 0
CA , 02/01 , 0
CA , 1991 , 0

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

1 { 99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
2 { 99504, 01/01/1993, 70in

AK , 01/01 , 304.8
AK , 1992 , 304.8
AK , 03/01 , 30.5
AK , 1992 , 30.5
AK , 01/01 , 21336
AK , 1993 , 21336

AK , 01/01 , [304.8, 21336]
AK , 03/01 , [30.5]
AK , 1992 , [304.8 , 30.5]
AK , 1993 , [21336]

AK , 01/01 , 21031
AK , 03/01 , 0
AK , 1992 , 274.3
AK , 1993 , 0

Run 2

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

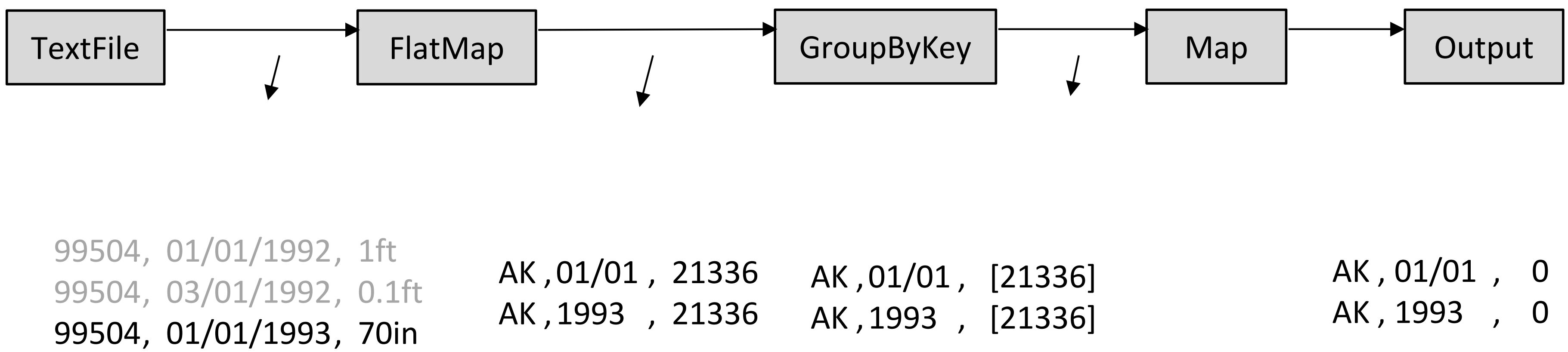
99504, 01/01/1992, 1ft	AK ,01/01 , 304.8	AK ,01/01 , [304.8]	AK , 01/01 , 0
99504, 03/01/1992, 0.1ft	AK ,1992 , 304.8	AK ,03/01 , [30.5]	AK , 03/01 , 0
99504, 01/01/1993, 70in	AK ,03/01 , 30.5	AK ,1992 , [304.8 , 30.5]	AK , 1992 , 274.3
	AK ,1992 , 30.5		

Run 3

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function



Run 4

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

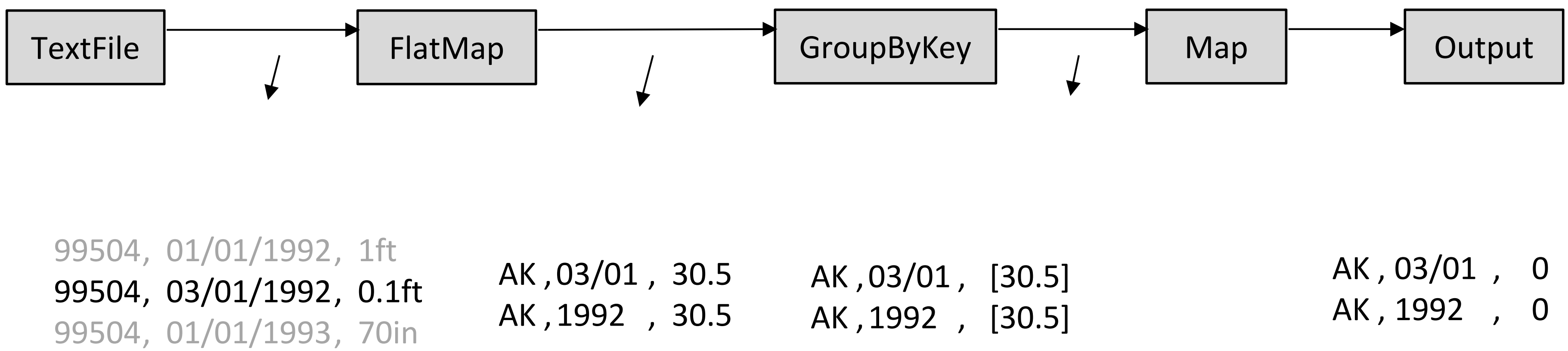
TextFile	FlatMap	GroupByKey	Output
99504, 01/01/1992, 1ft	AK ,01/01 , 304.8	AK ,01/01 , [304.8]	AK , 01/01 , 0
99504, 03/01/1992, 0.1ft	AK ,1992 , 304.8	AK ,1992 , [304.8]	AK , 1992 , 0
99504, 01/01/1993, 70in			

Run 5

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function



Run 6

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
99504, 01/01/1993, 70in

AK , 01/01 , 21336
AK , 1993 , 21336

AK , 01/01 , [21336]
AK , 1993 , [21336]

AK , 01/01 , 0
AK , 1993 , 0

Run 7

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

99504, 01/01/1992, 1ft	AK ,03/01 , 30.5	AK ,01/01 , [21336]	AK , 01/01 , 0
99504, 03/01/1992, 0.1ft	AK ,1992 , 30.5	AK ,03/01 , [30.5]	AK , 03/01 , 0
99504, 01/01/1993, 70in	AK ,01/01 , 21336	AK ,1992 , [30.5]	AK , 1992 , 0
	AK ,1993 , 21336	AK ,1993 , [21336]	AK , 1993 , 0

Run 8

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Existing Approach 2: Delta Debugging

- Delta Debugging performs a systematic binary-like search on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

99504, 01/01/1992, 1ft
99504, 03/01/1992, 0.1ft
99504, 01/01/1993, 70in

AK , 01/01 , 304.8
AK , 1992 , 304.8
AK , 01/01 , 21336
AK , 1993 , 21336

AK , 01/01 , [304.8, 21336]
AK , 1992 , [304.8]
AK , 1993 , [21336]

AK , 01/01 , 21031
AK , 1992 , 0
AK , 1993 , 0

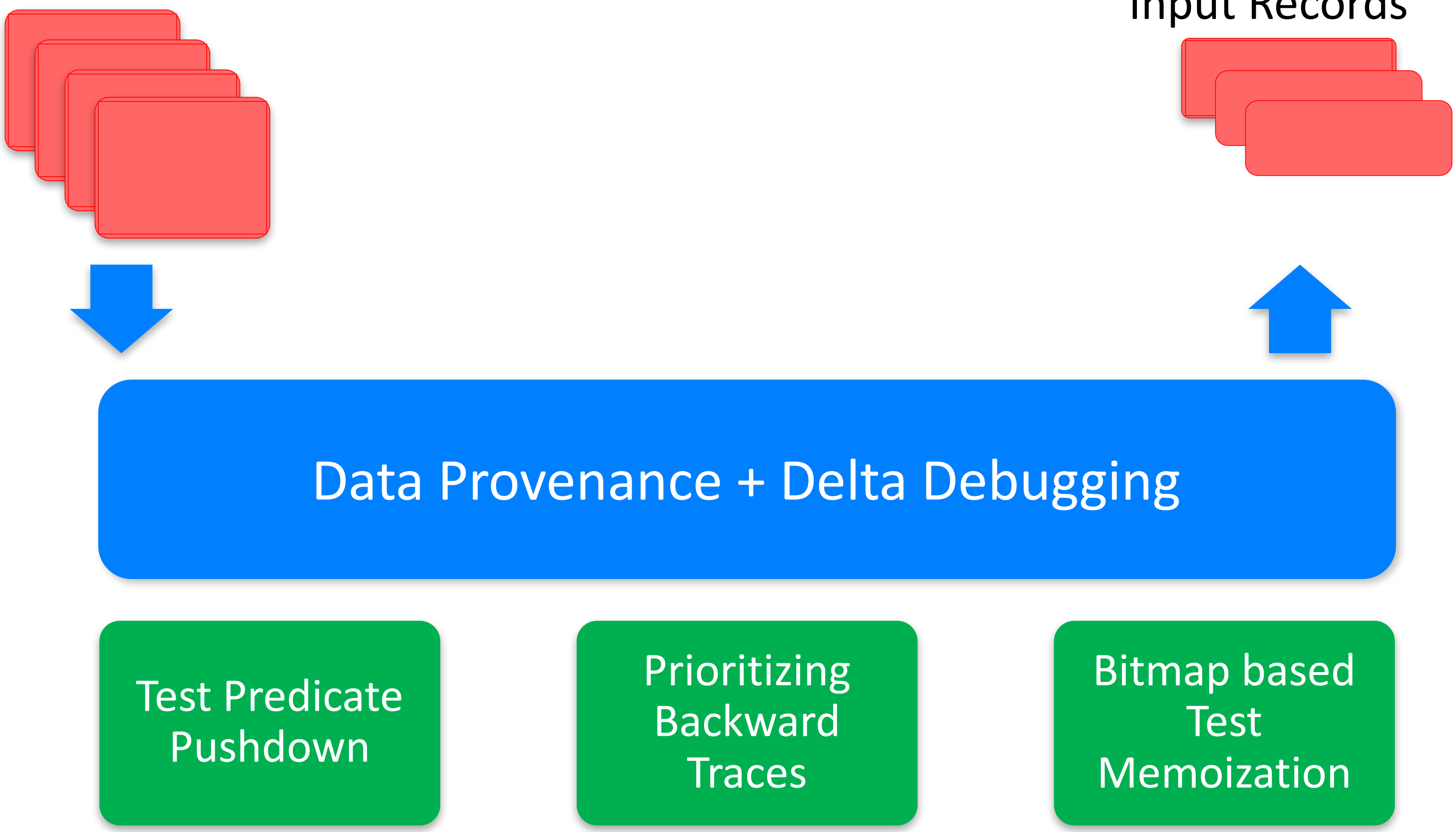
Run 9

It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

Automated Debugging in DISC with BigSift

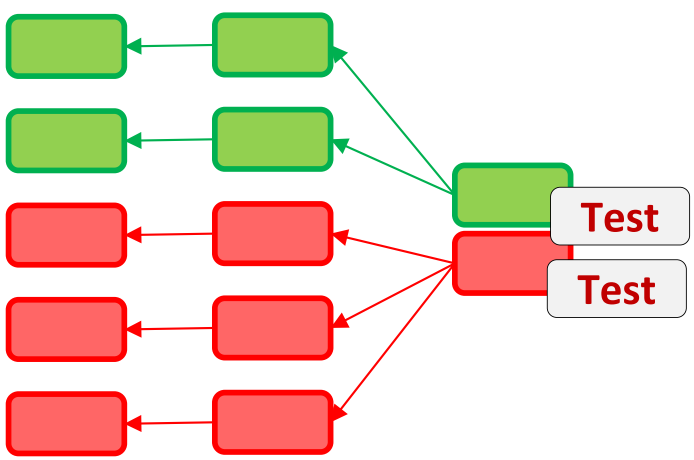
Input: A Spark Program, A Test Function

Output: Minimum Fault-Inducing Input Records

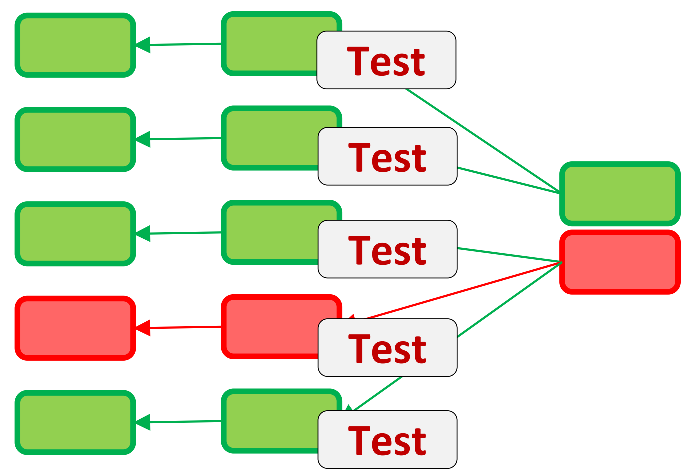


Optimization 1: Test Predicate Pushdown

- **Observation:** During backward tracing, data provenance traces through all partitions even though only a few partitions contain faulty intermediate data.



Without Test Pushdown

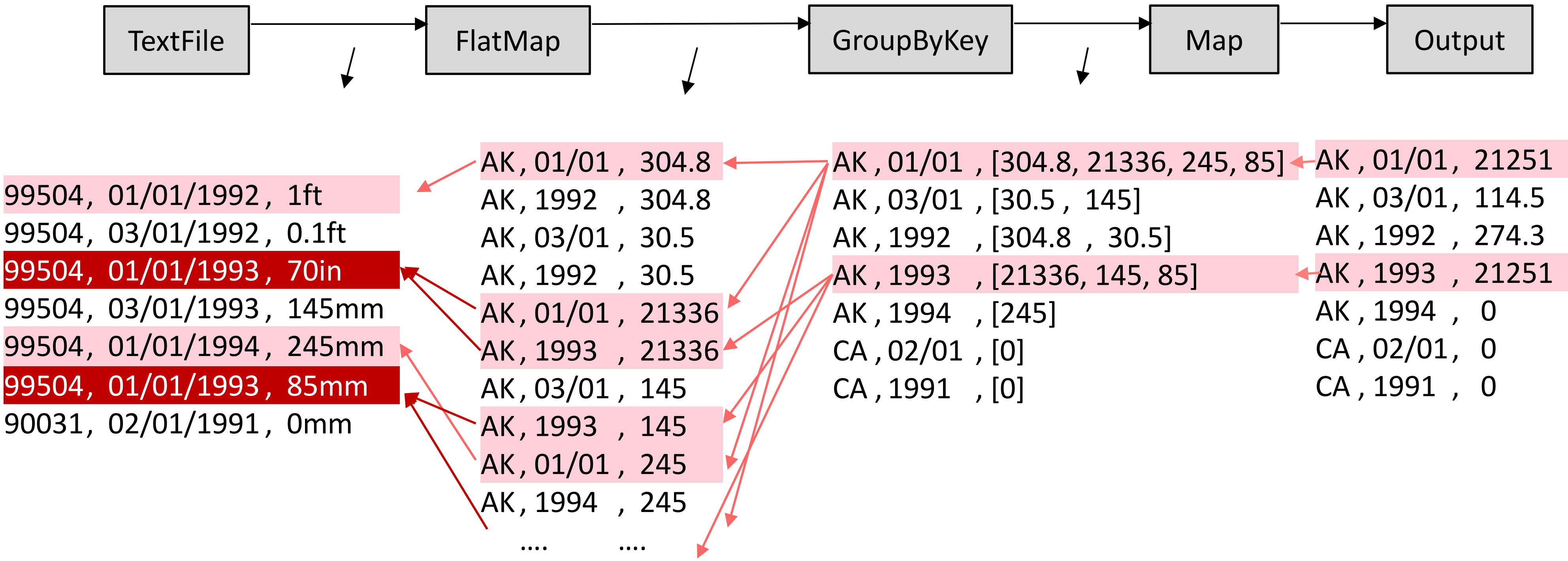


With Test Pushdown

If applicable, BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.

Optimization 2: Prioritizing Backward Traces

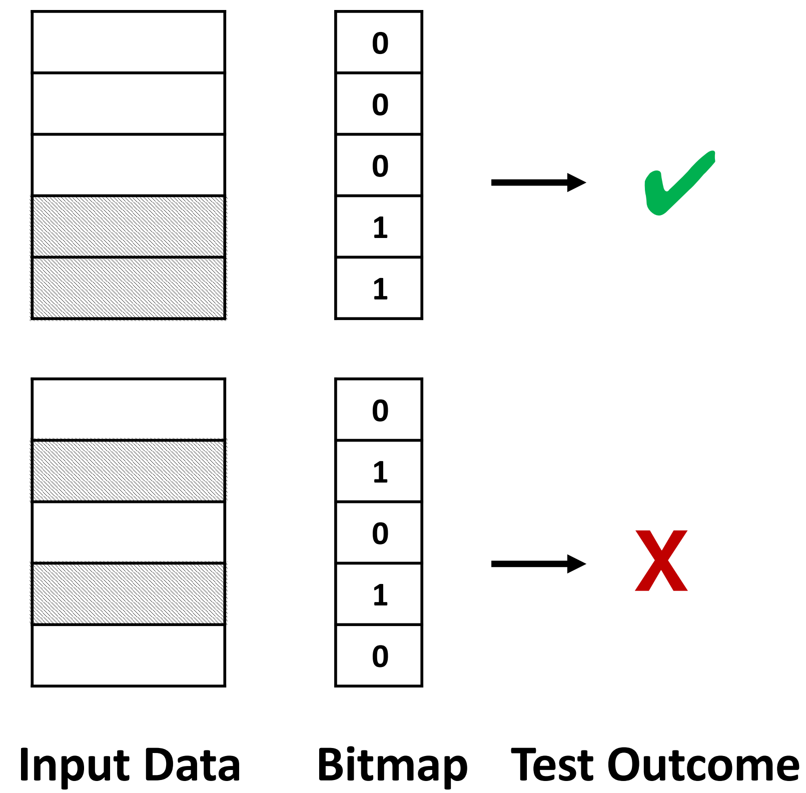
- **Observation:** The same faulty input record may contribute to multiple faulty output due to operators such as Join or Flatmap



In case of multiple faulty outputs, BigSift overlaps two backward traces to minimize the scope of fault-inducing input records

Optimization 3: Bitmap Based Test Memoization

- **Observation:** Delta debugging may try running a program on the same subset of input redundantly.
- BigSift leverages bitmap to compactly encode the offsets of original input to refer to an input subset



We use a bitmap based test memoization technique to avoid redundant testing of the same input dataset.

Evaluation Questions

- **RQ1:** How much improvement in the debugging time does BigSift provide in comparison to delta debugging?
- **RQ2:** How long is the debugging time of BigSift in comparison to original running time of a job?
- **RQ3:** How much improvement in the precision of fault-inducing input records does BigSift provide in comparison to data provenance?

RQ1: Performance Improvement over Delta Debugging

Subject Program		Running Time (sec)	Debugging Time (sec)		
Subject Program	Fault	Original Job	DD	BigSift	Improvement
Movie Histogram	Code	56.2	232.8	17.3	13.5X
Inverted Index	Code	107.7	584.2	13.4	43.6X
Rating Histogram	Code	40.3	263.4	16.6	15.9X
Sequence Count	Code	356.0	13772.1	208.8	66.0X
Rating Frequency	Code	77.5	437.9	14.9	29.5X
College Student	Data	53.1	235.3	31.8	7.4X
Weather Analysis	Data	238.5	999.1	89.9	11.1X
Transit Analysis	Code	45.5	375.8	20.2	18.6X

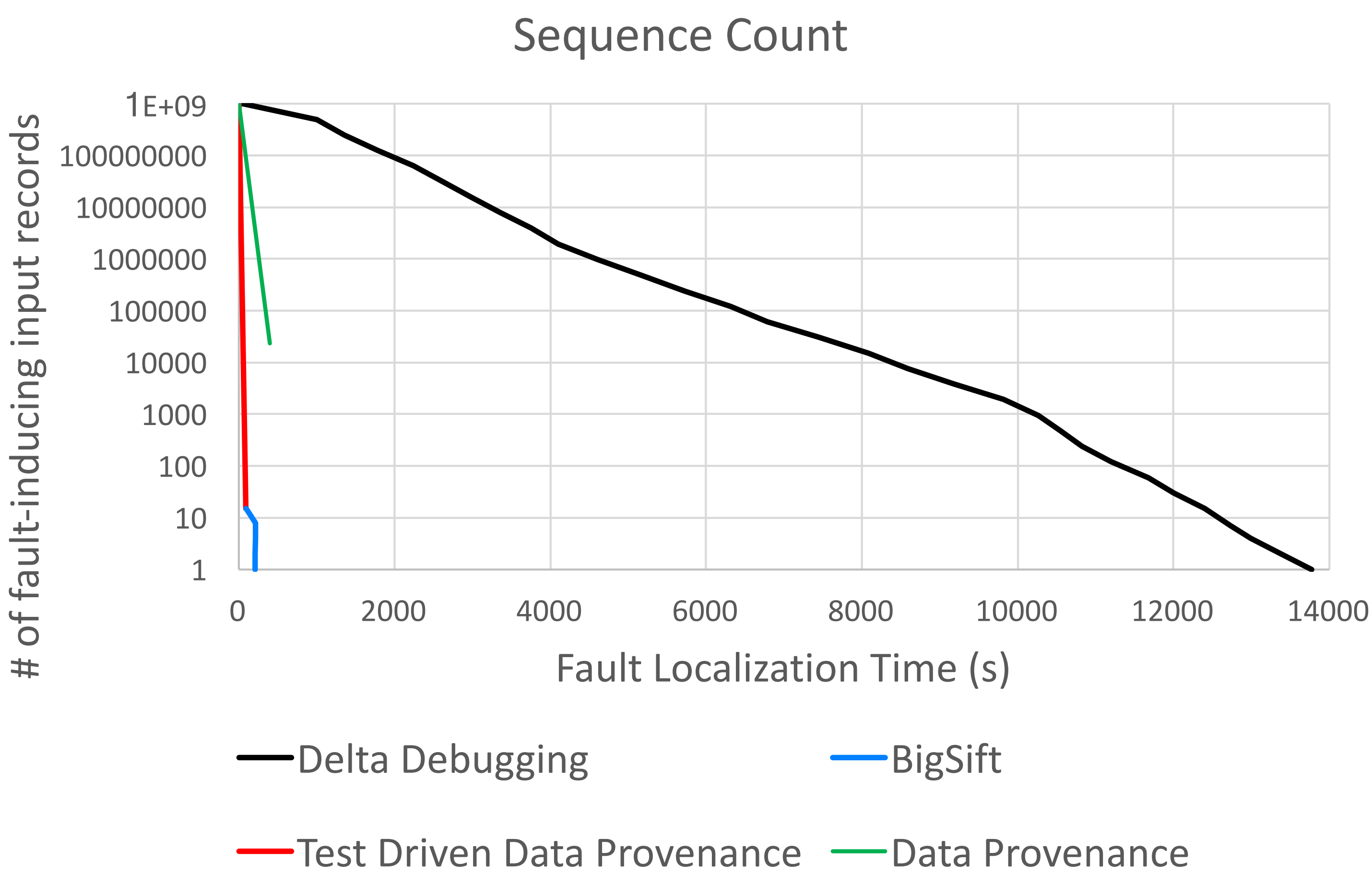
BigSift provides up to a 66X speed up in isolating the precise fault-inducing input records, in comparison to the baseline DD

RQ2: Debugging Time vs. Original Job Time

Subject Program		Running Time (sec)	Debugging Time (sec)		
Subject Program	Fault	Original Job	DD	BigSift	Improvement
Movie Histogram	Code	56.2	232.8	17.3	13.5X
Inverted Index	Code	107.7	584.2	13.4	43.6X
Rating Histogram	Code	40.3	263.4	16.6	15.9X
Sequence Count	Code	356.0	13772.1	208.8	66.0X
Rating Frequency	Code	77.5	437.9	14.9	29.5X
College Student	Data	53.1	235.3	31.8	7.4X
Weather Analysis	Data	238.5	999.1	89.9	11.1X
Transit Analysis	Code	45.5	375.8	20.2	18.6X

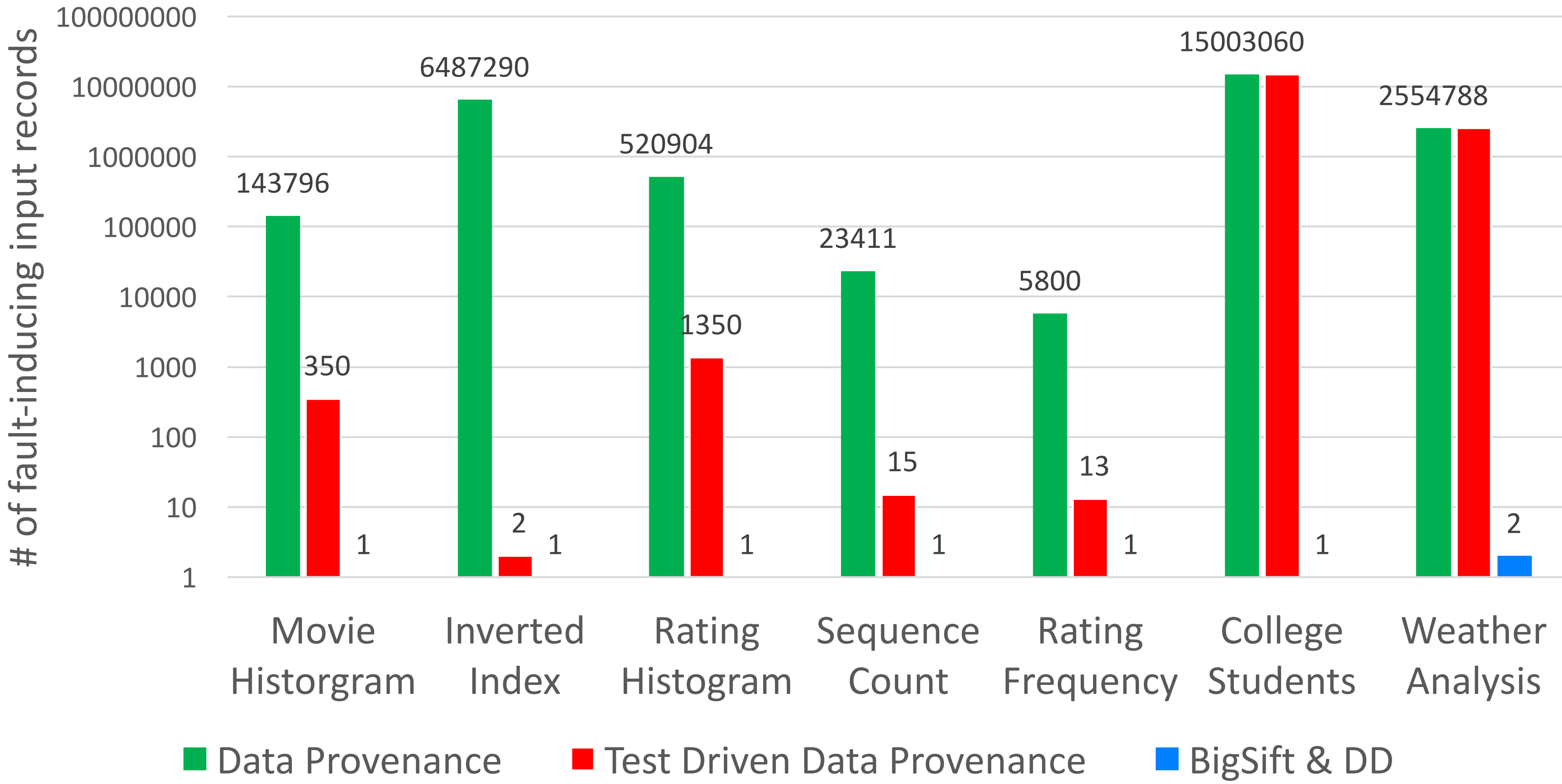
On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

RQ2: Debugging Time vs. Original Job Time



On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

RQ3: Fault Localizability over Data Provenance



BigSift leverages DD after DP to continue fault isolation, achieving several orders of magnitude 10^3 to 10^7 better precision.

Conclusion

- BigSift is the first piece of work in automated debugging of big data analytics in DISC.
- BigSift provides **10^3X – 10^7X more precision** than data provenance in terms of fault localizability.
- It provides up to **66X speed up** in debugging time over baseline Delta Debugging.
- In our evaluation we have observed that, on average, BigSift finds the faulty input in **62% less** than the original job execution time.

Questions?