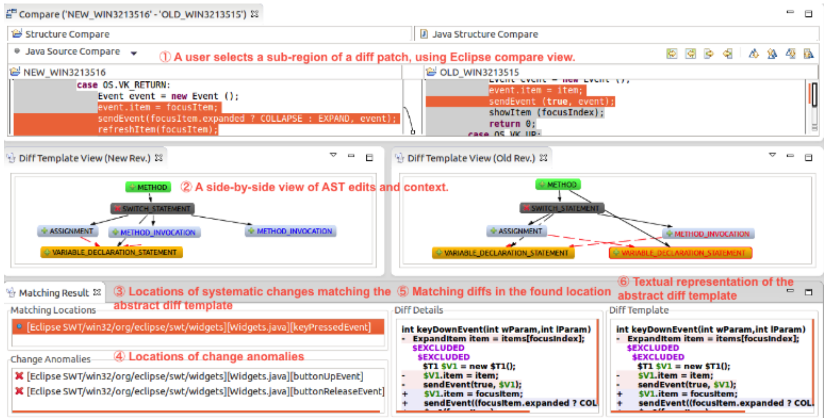
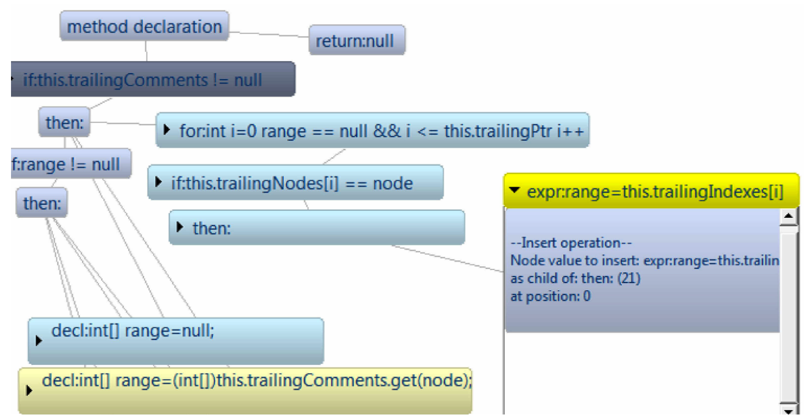
Interactive and Automated Debugging for Big Data Analytics

Miryung Kim
University of California, Los Angeles

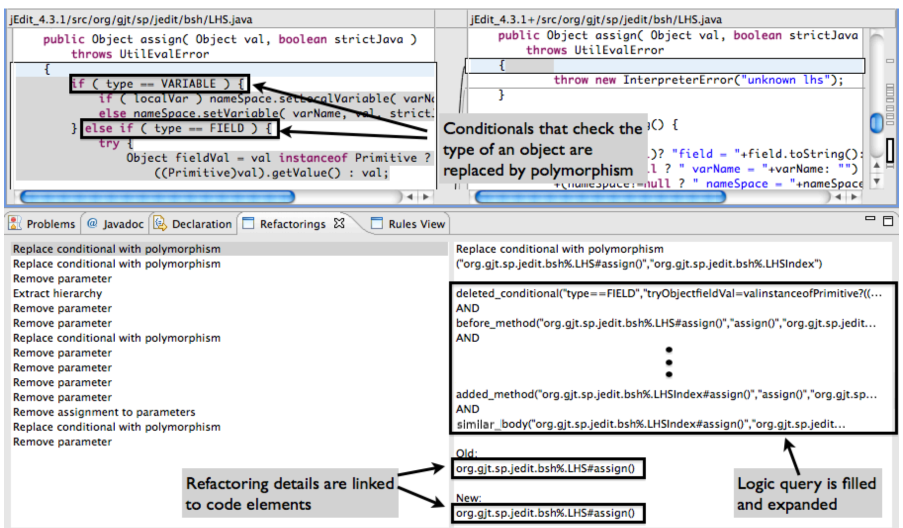
Software Engineering and Analysis Lab at UCLA



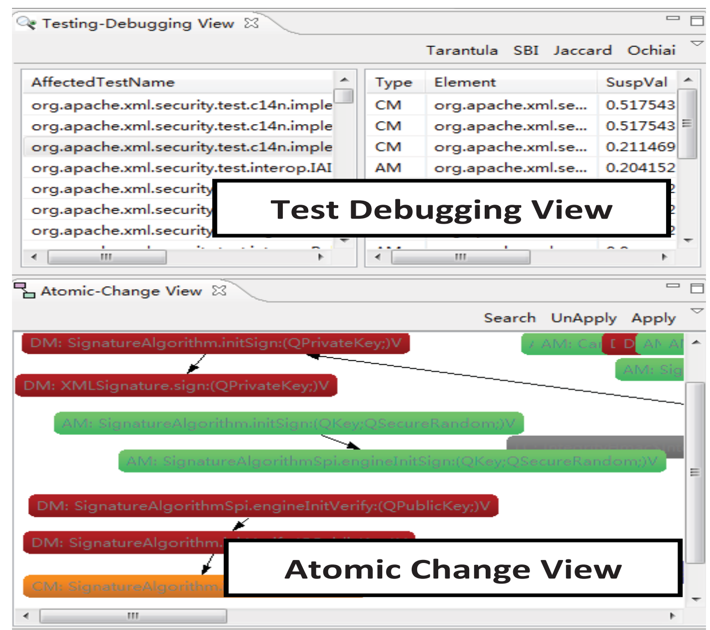
Interactive Code Review



Refactoring and Transformation

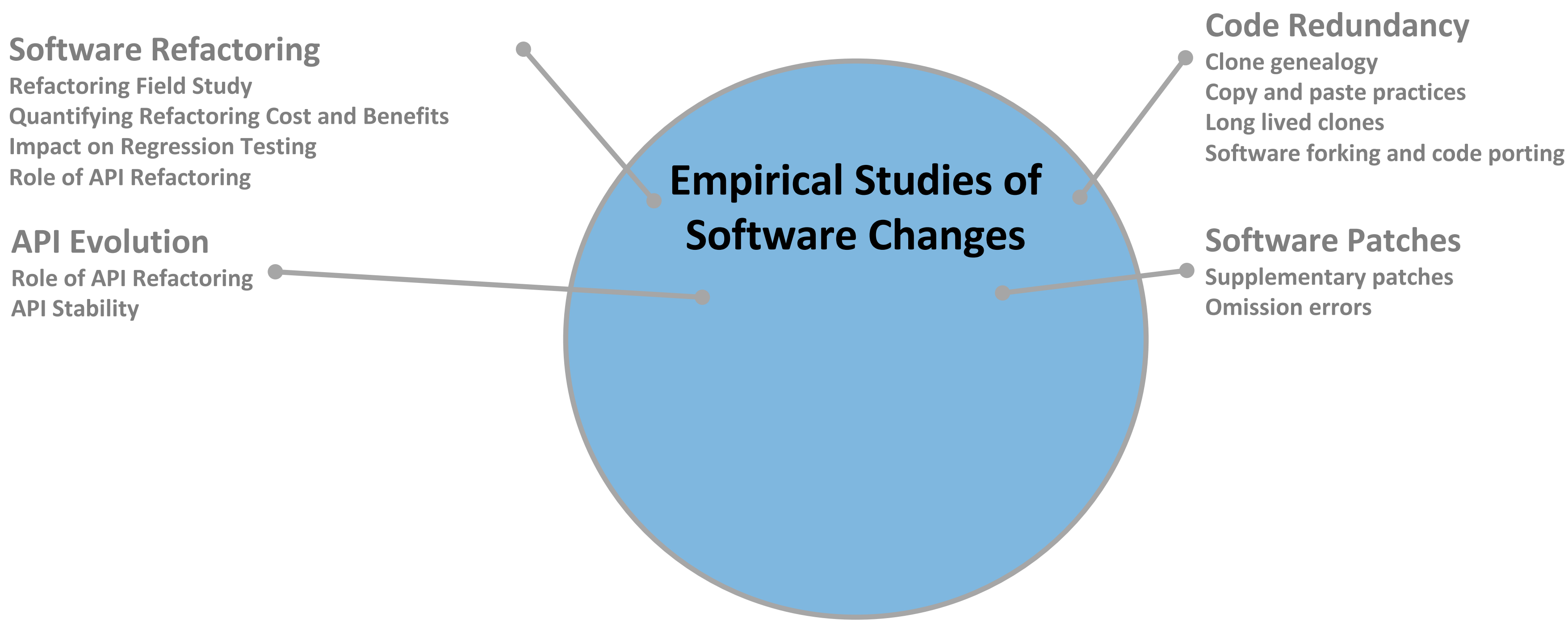


Program Comprehension

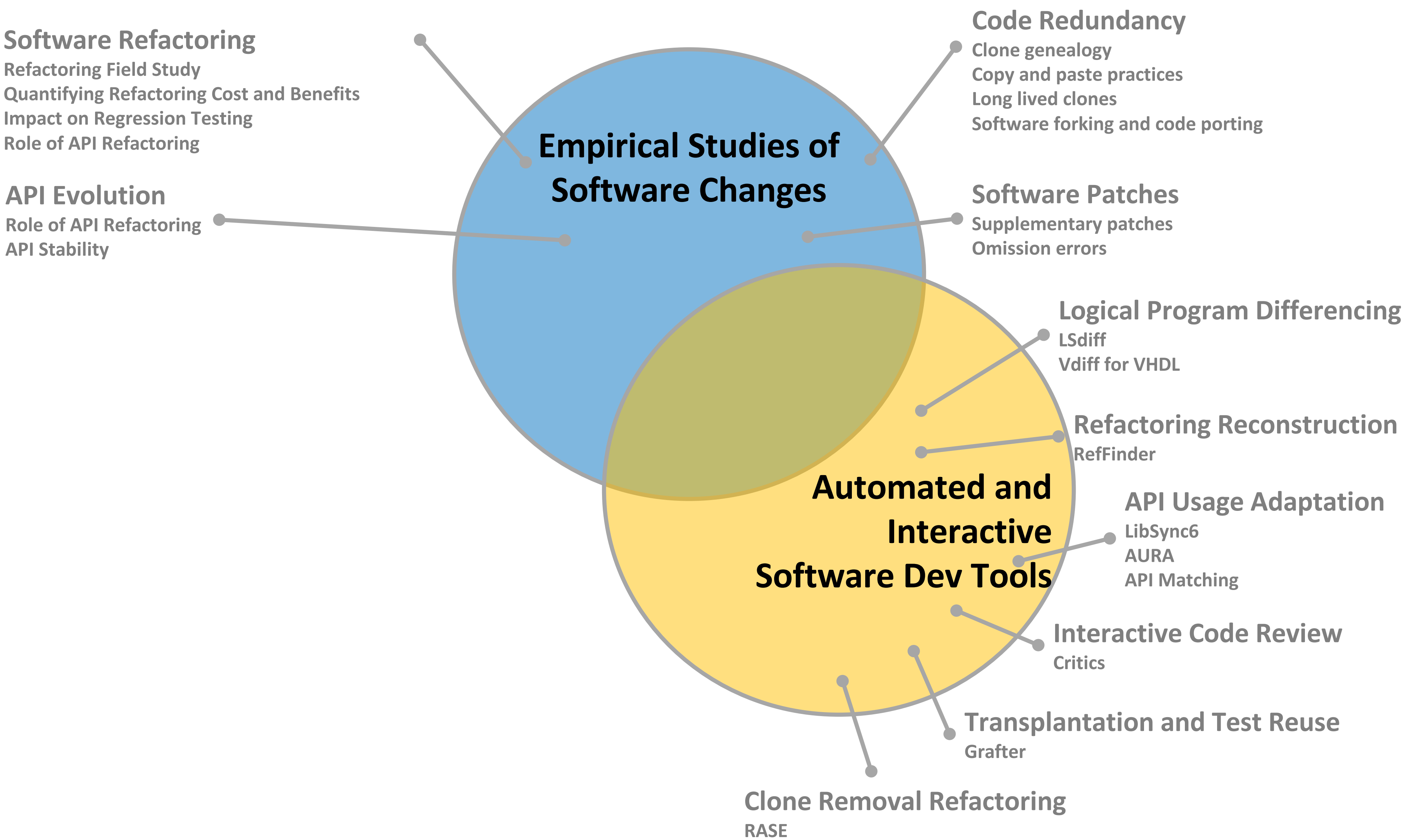


Debugging

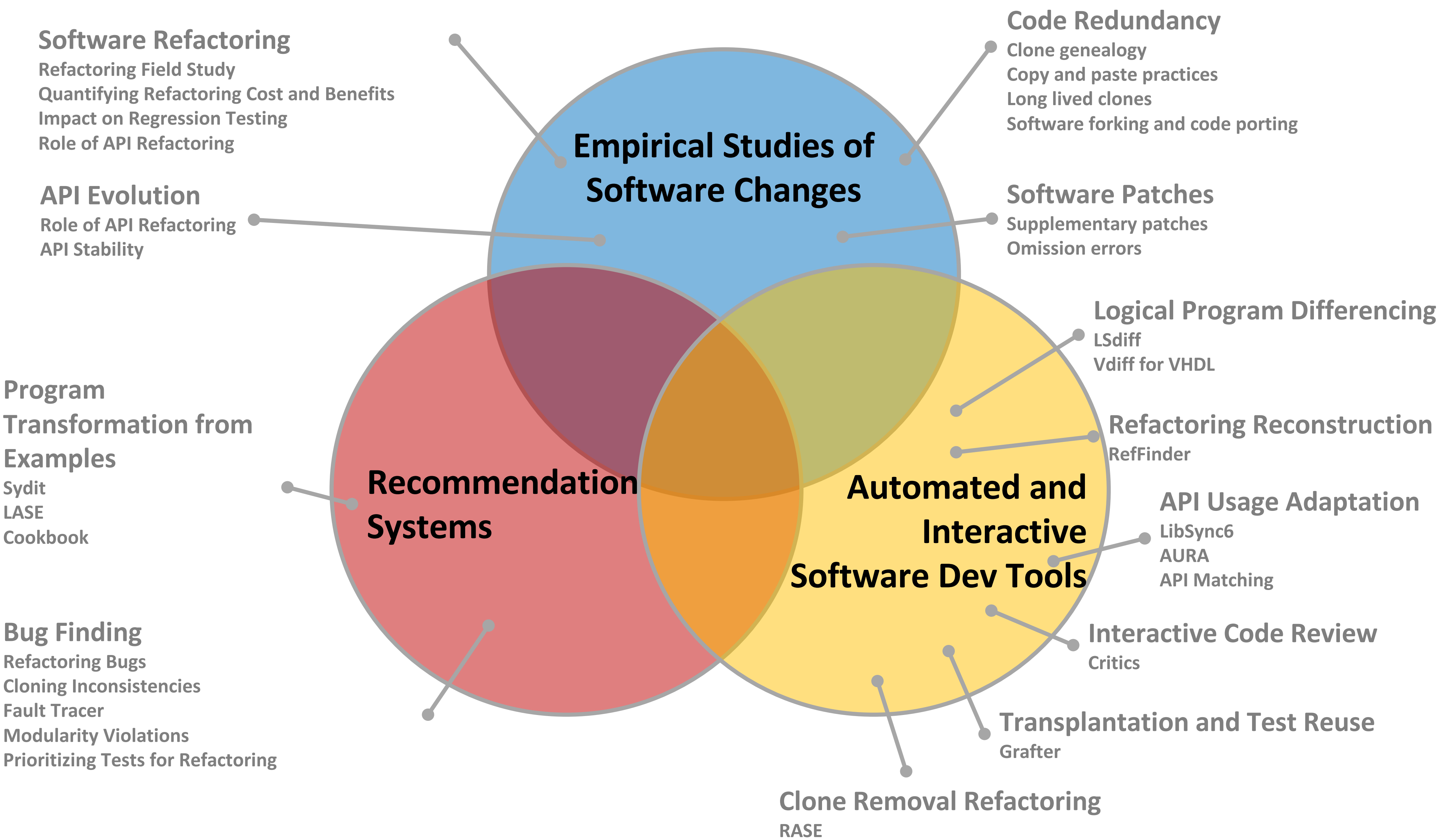
Data Science *elevating* Software Engineering



Data Science *elevating* Software Engineering



Data Science *elevating* Software Engineering



Current Research Focus:
Software Engineering *elevating* Data Science

Data Scientists in Software Teams

- Background
- Work Activities
- Challenges
- Best Practices
- Quality Assurance

SE Tools for Big Data Analytics

- Interactive Debugger
- Data Provenance
- Automated Debugging

The Emerging Roles of Data Scientists on Software Teams [ICSE 2016]

We are at a **tipping point** where there are large scale telemetry, machine, process and quality data.

Data scientists are emerging roles in SW teams due to an increasing demand for **experimenting with real users** and reporting results with statistical rigor.

We have conducted **the first in-depth interview study** and **the largest scale survey** of **professional data scientists** to characterize working styles.

Insight Provider

Specialists

Platform Builder

Polymath

Team Leader

Methodology for Studying "Data Scientists"

In-Depth Interviews [ICSE 2016]

16 data scientists

- 5 women and 11 men from eight different Microsoft organizations

Snowball sampling

- data-driven engineering meet-ups and technical community meetings
- word of mouth

Coding with Atlas.TI

Clustering of participants

Survey [TSE & ICSE 2018 Journal First]

793 responses

- full-time data scientists
- employees with interest in data science

Questions about

- demographics
- skills and tool usage
- self-perception
- working styles
- time spent
- challenges and best practices

Background of Data Scientists

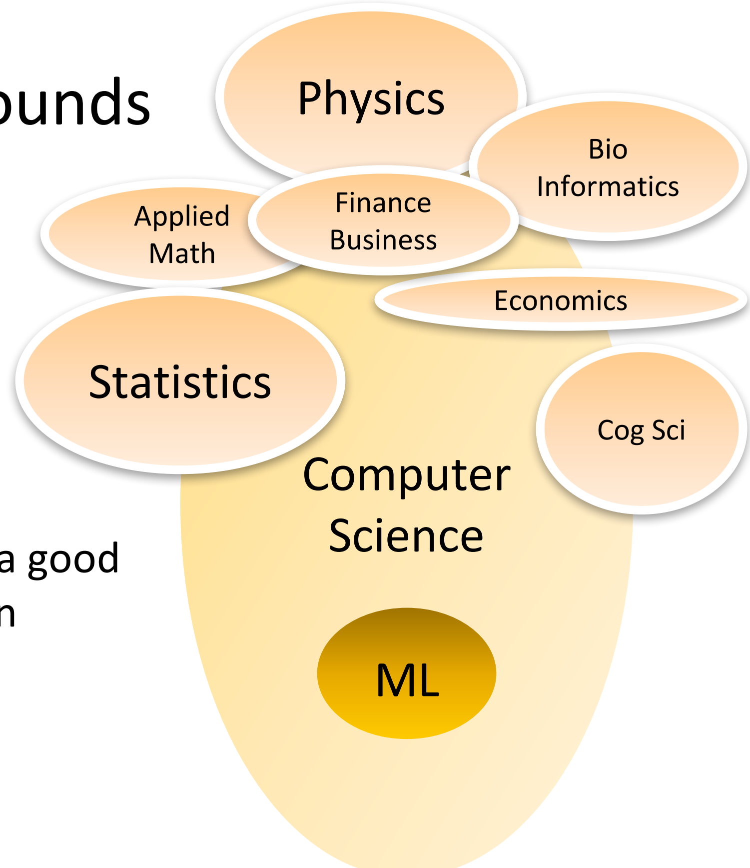
Most CS, many **interdisciplinary** backgrounds

Many have **higher education** degrees

Survey: 41% have master's degrees, and 22% have PhDs

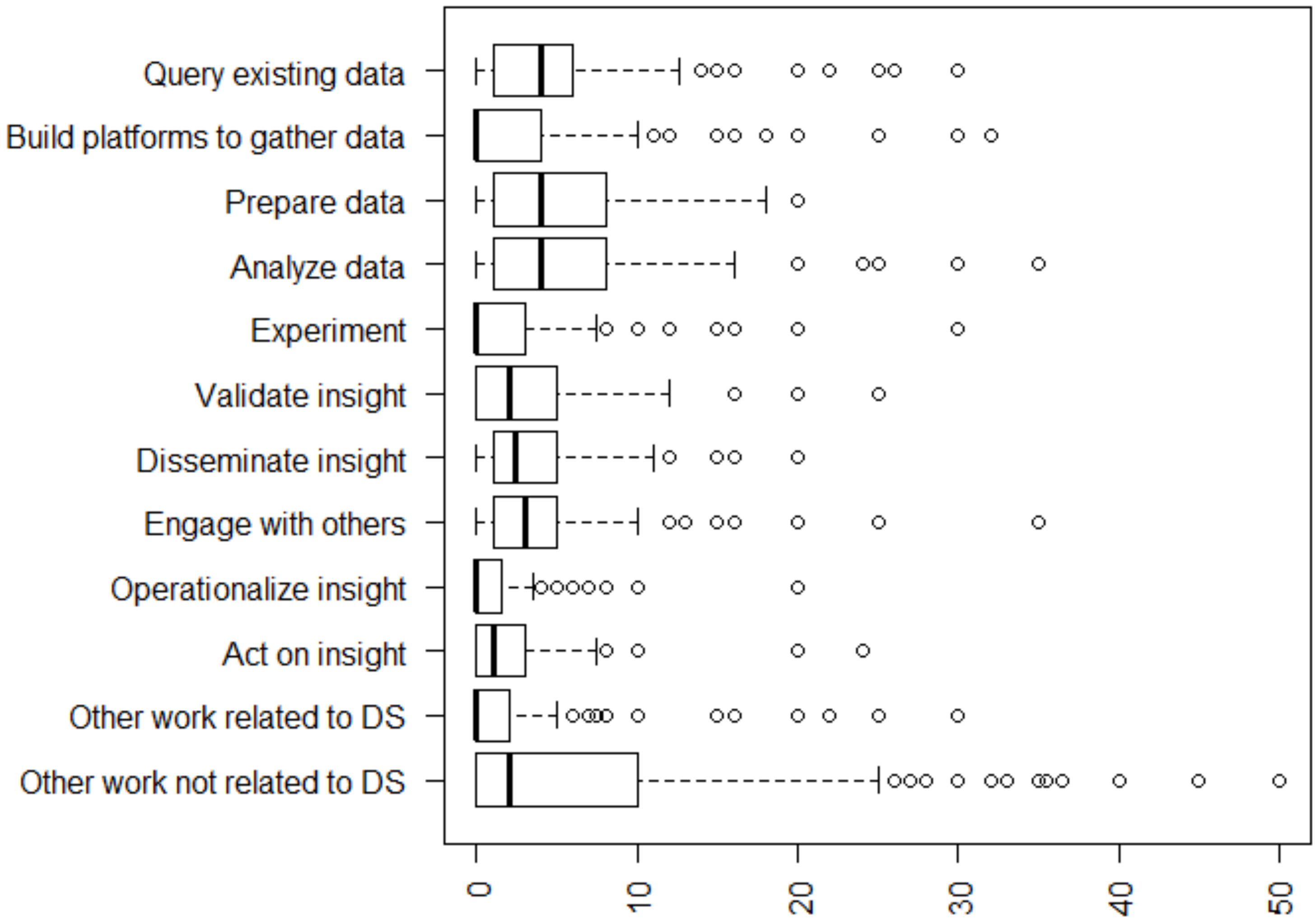
PhD training contributes to working style

"Doing data science is kind of like doing research. It looks like a good problem and looks like a good idea. You think you may have an approach, but then maybe you end up with a dead end." [P5]



Time Spent on Activities

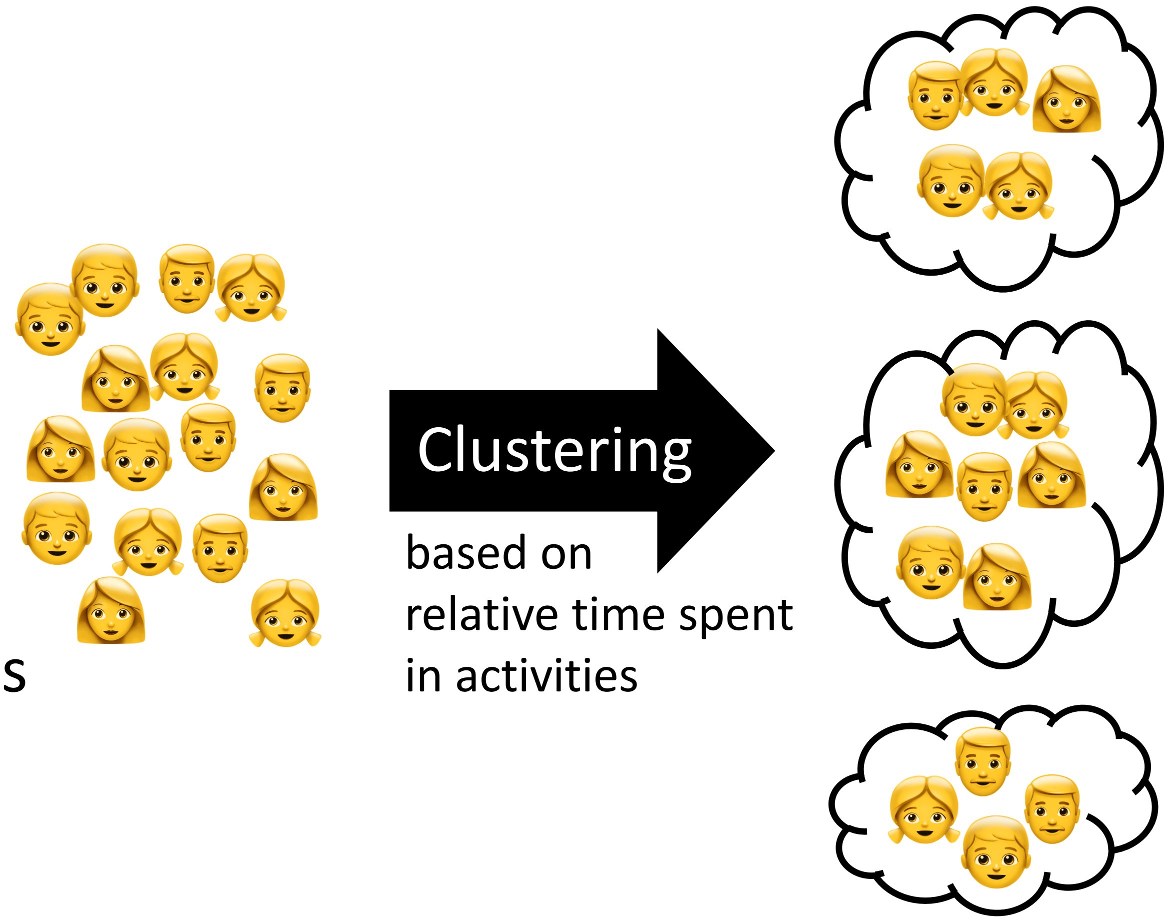
Hours spent on certain activities (self reported, survey, N=532)



Time Spent on Activities

Cluster analysis on relative time spent (k-means)

532 data scientists
at Microsoft



9 Distinct Categories of Data Scientists based on Work Activities

←Clusters / Activities →

Clusters	Query existing data	Build platforms to gather data	Prepare data	Analyze data	Experiment	Validate insight	Disseminate insight	Engage with others	Operationalize insight	Act on insight	Other work related to DS	Other work not related to DS
Entire population 532 people	12.0% 4.7h	7.2% 2.9h	11.7% 4.9h	12.5% 5.2h	4.8% 2.1h	6.9% 3.0h	8.5% 3.5h	9.2% 3.8h	2.4% 1.1h	5.5% 2.1h	4.1% 1.9h	15.1% 6.7h
Cluster 1 Polymath 156 people	10.4% 4.4h	8.5% 3.6h	11.5% 5.1h	15.1% 6.7h	9.1% 4.0h	7.7% 3.6h	7.4% 3.5h	7.9% 3.6h	3.2% 1.5h	5.2% 2.3h	4.0% 2.0h	10.1% 4.5h
Cluster 2 Data Evangelist 71 people	6.8% 2.2h	2.1% 1.0h	6.7% 2.5h	7.7% 2.9h	2.4% 1.2h	7.0% 2.6h	12.0% 4.5h	23.0% 8.6h	3.7% 1.3h	9.5% 3.3h	13.4% 6.0h	5.7% 2.6h
Cluster 3 Data Preparer 122 people	24.5% 9.4h	4.9% 1.9h	19.6% 7.8h	10.0% 4.0h	3.0% 1.3h	9.0% 4.1h	11.6% 4.5h	8.8% 3.5h	1.5% 0.7h	3.9% 1.3h	1.5% 0.7h	1.8% 0.8h
Cluster 4 Data Shaper 33 people	5.6% 2.5h	1.8% 0.7h	27.0% 11.5h	25.7% 10.9h	6.0% 2.6h	8.9% 3.8h	7.6% 3.3h	7.5% 3.2h	2.1% 1.0h	3.3% 1.4h	2.5% 1.1h	1.9% 0.9h
Cluster 5 Data Analyzer 24 people	9.9% 3.7h	0.9% 0.3h	5.8% 2.4h	49.1% 18.4h	4.6% 2.2h	6.6% 2.7h	5.2% 2.2h	5.8% 2.4h	1.8% 0.9h	4.2% 1.6h	2.8% 1.3h	3.2% 1.3h
Cluster 6 Platform Builder 27 people	12.5% 4.4h	48.5% 18.4h	6.1% 2.6h	4.3% 1.9h	3.8% 1.1h	2.7% 1.2h	4.4% 2.0h	4.1% 1.9h	2.1% 0.9h	3.0% 1.1h	1.4% 0.6h	6.9% 3.1h
Cluster 7 Moonlighter 50% 63 people	7.3% 3.1h	5.0% 2.2h	5.0% 2.1h	5.5% 2.4h	2.8% 1.2h	4.2% 2.0h	7.8% 3.3h	5.9% 2.4h	1.8% 0.8h	5.7% 2.3h	2.5% 1.1h	46.5% 20.0h
Cluster 8 Moonlighter 10% 32 people	2.9% 1.2h	1.4% 0.6h	1.9% 0.9h	1.6% 0.7h	0.4% 0.2h	1.5% 0.7h	1.7% 0.8h	2.3% 1.0h	0.6% 0.3h	2.1% 1.0h	2.9% 1.3h	80.9% 36.1h
Cluster 9 Act on Insight 4 people	0.9% 0.1h	2.1% 1.0h	1.8% 0.2h		0.9% 0.1h	5.7% 1.5h	18.5% 4.8h	10.1% 1.6h	3.0% 1.1h	57.1% 11.8h		

Data Scientists in Software Teams:
State of the Art and Challenges, Kim et al.
IEEE Transactions on Software Engineering

Category 1: Data Shaper

	Query existing data	Build platforms to gather data	Prepare Data	Analyze Data	Experiment	Validate insight	Disseminate insight	Engage with others	Operationalize insight	Act on insight	Other work related to DS	Other work not related to DS	
Entire population 532 people	12.0% 4.7h	7.2% 2.9h	11.7% 4.9h	12.5% 5.2h	4.8% 2.1h	6.9% 3.0h	8.5% 3.5h	9.2% 3.8h	2.4% 1.1h	5.5% 2.1h	4.1% 1.9h	15.1% 6.7h	Entire Population
Cluster 4 Data Shaper 33 people	5.6% 2.5h	1.8% 0.7h	27.0% 11.5h	25.7% 10.9h	6.0% 2.6h	8.9% 3.8h	7.6% 3.3h	7.5% 3.2h	2.1% 1.0h	3.3% 1.4h	2.5% 1.1h	1.9% 0.9h	Data Shaper

↑PhD Degree: 54% vs. 21%
↑Master's Degree: 88% vs. 61%
↑Algorithms: 71% vs. 46%
↑Machine Learning: 92% vs. 49%
↑Optimization: 42% vs. 19%

↓Structured Data: 46% vs. 69%
↓Front End Programming: 13% vs. 34%
↑MATLAB: 30% vs. 5%
↑Python: 48% vs. 22%
↑TLC: 35% vs. 11% ↓Excel: 57% vs. 84%

Category 2: Platform Builder

	Query existing data	Build platforms to gather data	Prepare data	Analyze data	Experiment	Validate insight	Disseminate insight	Engage with others	Operationalize insight	Act on insight	Other work related to DS	Other work not related to DS	
Entire population 532 people	12.0% 4.7h	7.2% 2.9h	11.7% 4.9h	12.5% 5.2h	4.8% 2.1h	6.9% 3.0h	8.5% 3.5h	9.2% 3.8h	2.4% 1.1h	5.5% 2.1h	4.1% 1.9h	15.1% 6.7h	Entire Population
Cluster 6 Platform Builder 27 people	12.5% 4.4h	48.5% 18.4h	6.1% 2.6h	4.3% 1.9h	3.8% 1.1h	2.7% 1.2h	4.4% 2.0h	4.1% 1.9h	2.1% 0.9h	3.0% 1.1h	1.4% 0.6h	6.9% 3.1h	Platform Builder

↑Back End Programming: 70% vs. 36%
↑Big and Distributed Data: 81% vs. 50%
↑Front End Programming: 63% vs. 31%
↑SQL: 89% vs. 68%

↑C/C++/C#: 70% vs. 45%
↓Classic Statistics: 30% vs. 50%

Category 3: Data Analyzer

	Query existing data	Build platforms to gather data	Prepare data	Analyze Data	Experiment	Validate insight	Disseminate insight	Engage with others	Operationalize insight	Act on insight	Other work related to DS	Other work not related to DS	
Entire population 532 people	12.0% 4.7h	7.2% 2.9h	11.7% 4.9h	12.5% 5.2h	4.8% 2.1h	6.9% 3.0h	8.5% 3.5h	9.2% 3.8h	2.4% 1.1h	5.5% 2.1h	4.1% 1.9h	15.1% 6.7h	Entire Population
Cluster 5 Data Analyzer 24 people	9.9% 3.7h	0.9% 0.3h	5.8% 2.4h	49.1% 18.4h	4.6% 2.2h	6.6% 2.7h	5.2% 2.2h	5.8% 2.4h	1.8% 0.9h	4.2% 1.6h	2.8% 1.3h	3.2% 1.3h	Data Analyzer

- ↑ Population "Data Science Employees"
- ↑ Master's degree: 82% vs. 61%
- ↓ Professional Experience: 8.4 yr vs. 14.3 yr
- ↓ Years at Microsoft: 3.7 yr vs. 7.4 yr
- ↑ Bayesian Monte Carlo Stat: 42% vs. 15%
- ↑ Classical Stats: 76% vs. 47%
- ↑ Data Manipulation: 82% vs. 54%
- ↓ Front End Programming: 12vs. 34%
- ↑ Math: 66% vs. 47%
- ↓ Product Development: 27% vs. 46%
- ↑ R: 64% vs. 38%
- ↓ Office BI: 15% vs. 37%

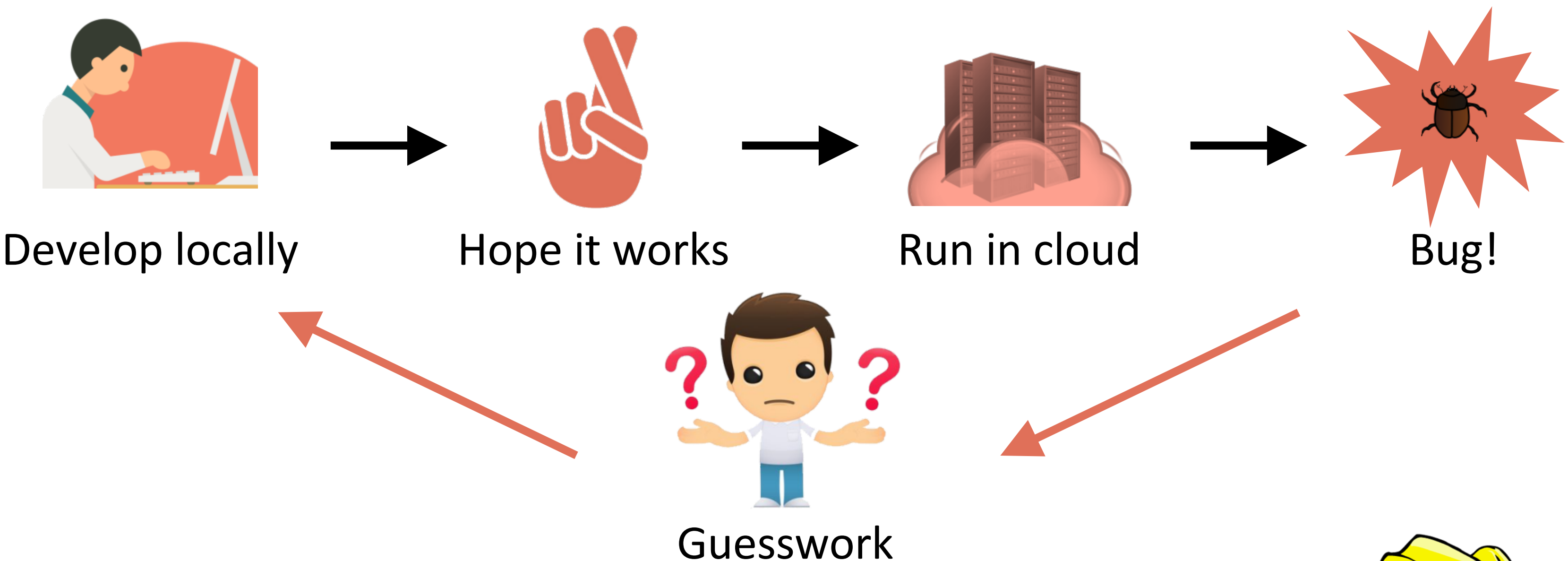
Challenges in Ensuring “Correctness”

Validation is a major challenge.

“Honestly, we don’t have a good method for this.” [P457]

“Just because the math is right, doesn’t mean that the answer is right.” [P307]

Explainability is important. Participants warned about overreliance on aggregate metrics— **“to gain insights, you must go one level deeper.”**



Google Map Reduce

hadoop

Spark

HIVE

Software Engineering for Data Science

Data Scientists in Software Teams

- Background
- Work Activities
- Challenges
- Best Practices
- Quality Assurance

SE Tools for Big Data Analytics

- Interactive Debugger [ICSE '16]
- Data Provenance [VLDB '16]
- Automated Debugging [SoCC '17]

Data Scientists | Big Data Debugging

BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark

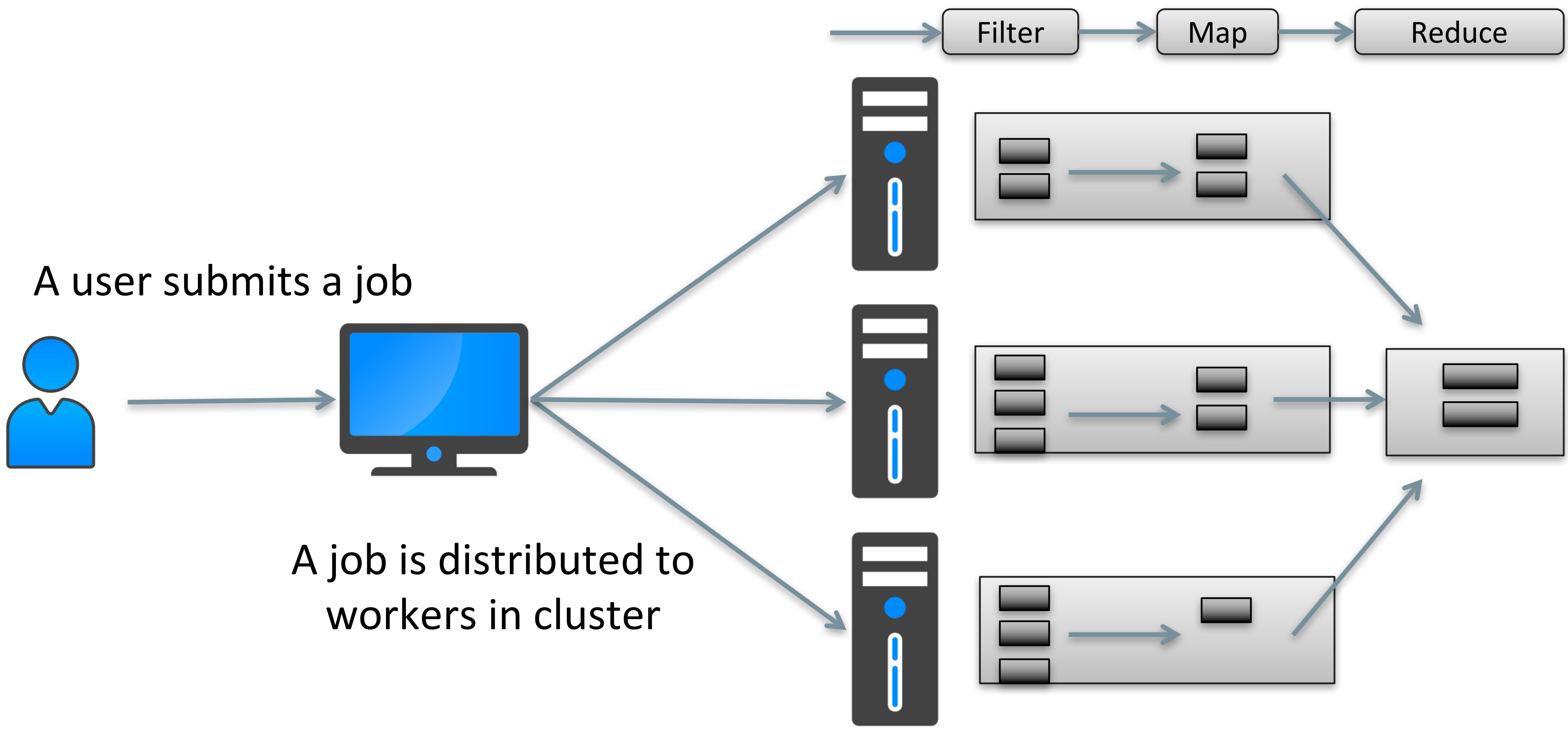
Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, Miryung Kim

[ICSE 2016, FSE Tool Demo 2016, SIGMOD Tool Demo 2017]

UCLA

Running a Map Reduce Job on Cluster

A user submits a job



A job is distributed to workers in cluster

Each worker performs pipelined transformations on a partition with millions of records

Motivating Scenario: Election Record Analysis

- Alice writes a Spark program that runs correctly on local machine (100MB data) but crashes on cluster (1TB)
- Alice cannot see the crash-inducing intermediate result.
- Alice cannot identify which input from 1TB causing crash
- When crash occurs, all intermediate results are thrown away.

VoterID	Candidate	State	Time
9213	Sanders	TX	1440023087

```
val log = "s3n://poll.log"
val text_file = spark.textFile(log)
val count = text_file
 .filter( line => line.split()[3].toInt
> 1440012701)
 .map(line = > (line.split()[1] , 1))
 .reduceByKey(_ + _).collect()
```

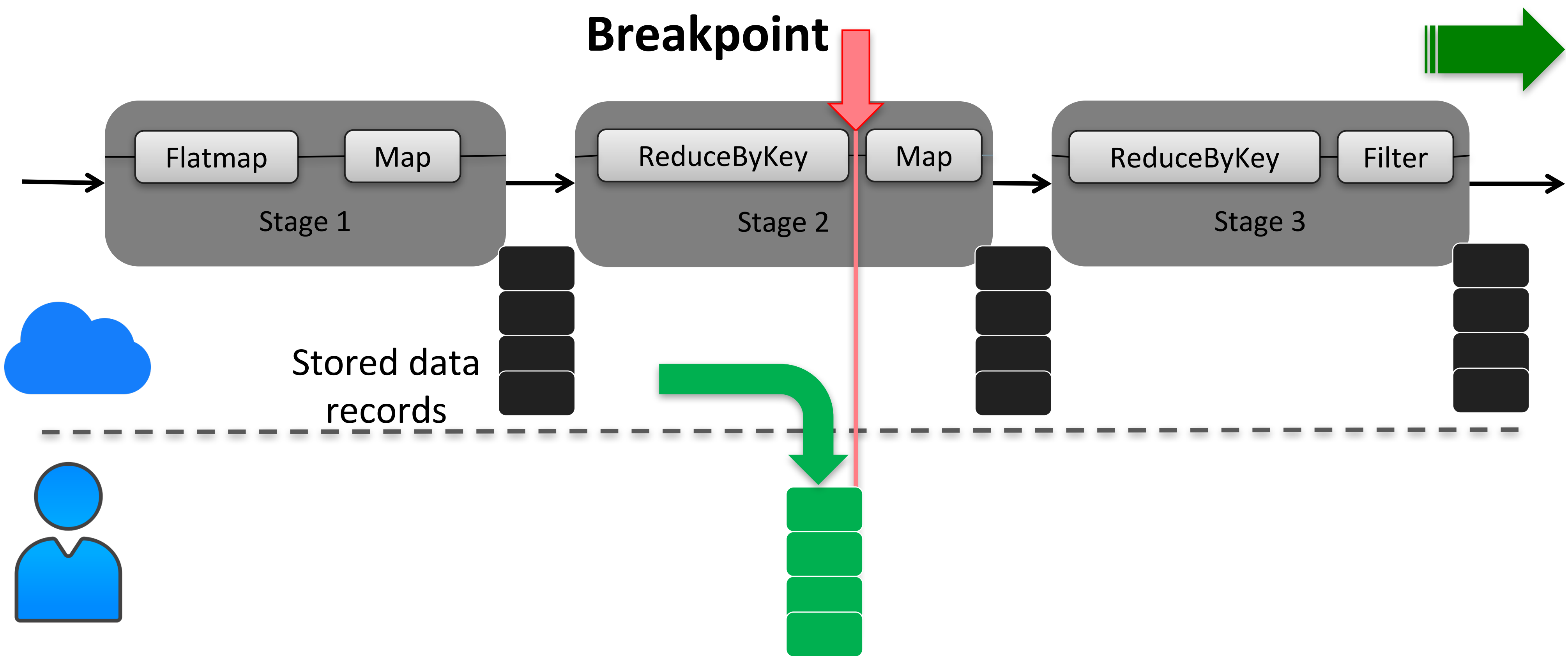
```
Task 31 failed 3 times; aborting
job
ERROR Executor: Exception in
task 31 in stage 0 (TID 31)

java.lang.NumberFormatException
```

Why Traditional Debug Primitives Do Not Work for Apache Spark?

Enabling interactive debugging requires us to **re-think the features of traditional debugger** such as GDB

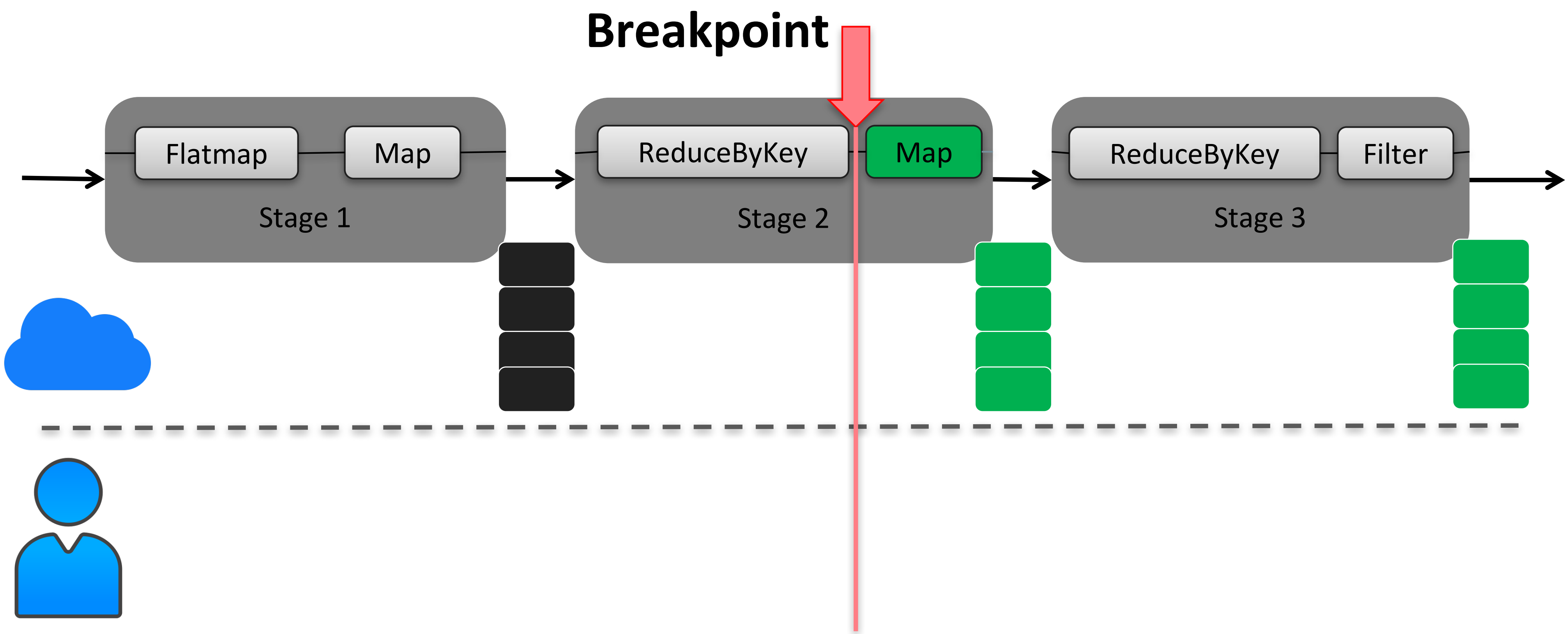
- Pausing the entire computation on the cloud could reduce throughput
- It is clearly infeasible for a user to inspect billion of records through a regular watchpoint
- Even launching remote JVM debuggers to individual worker nodes cannot scale for big data computing

1. Simulated Breakpoint



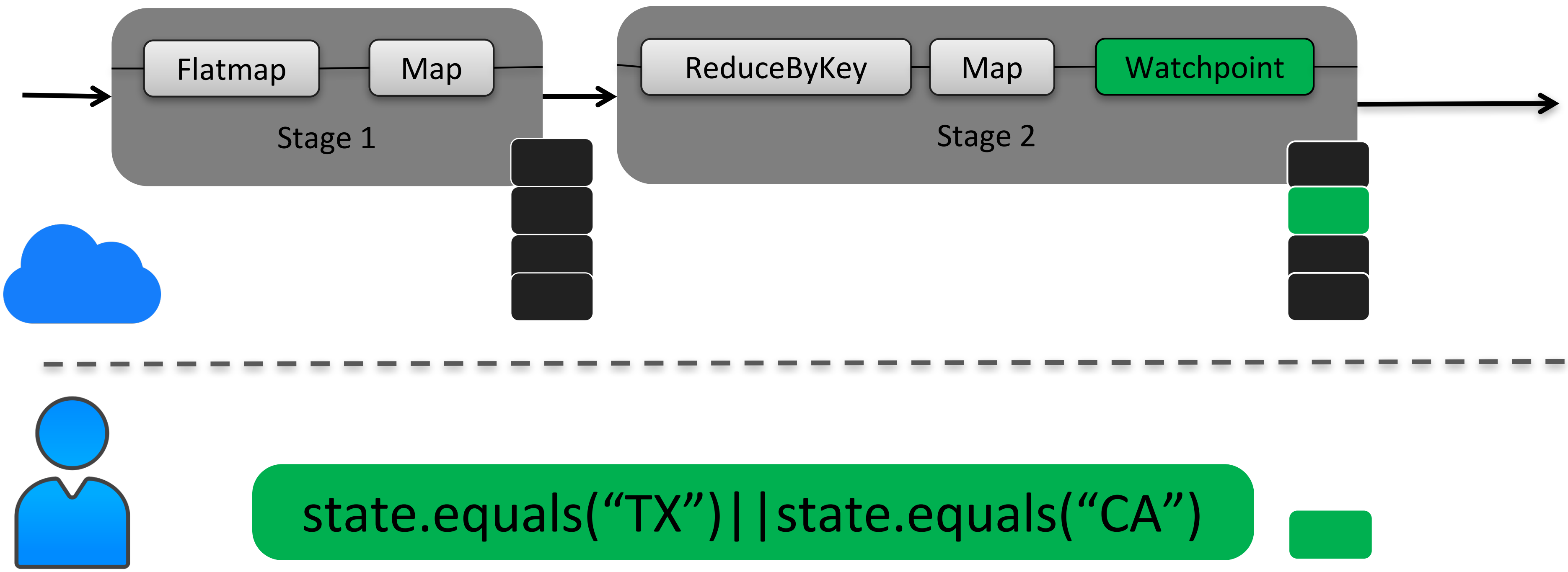
Simulated breakpoint replays computation from the latest materialization point where data is stored in memory

1. Simulated Breakpoint – Realtime Code Fix



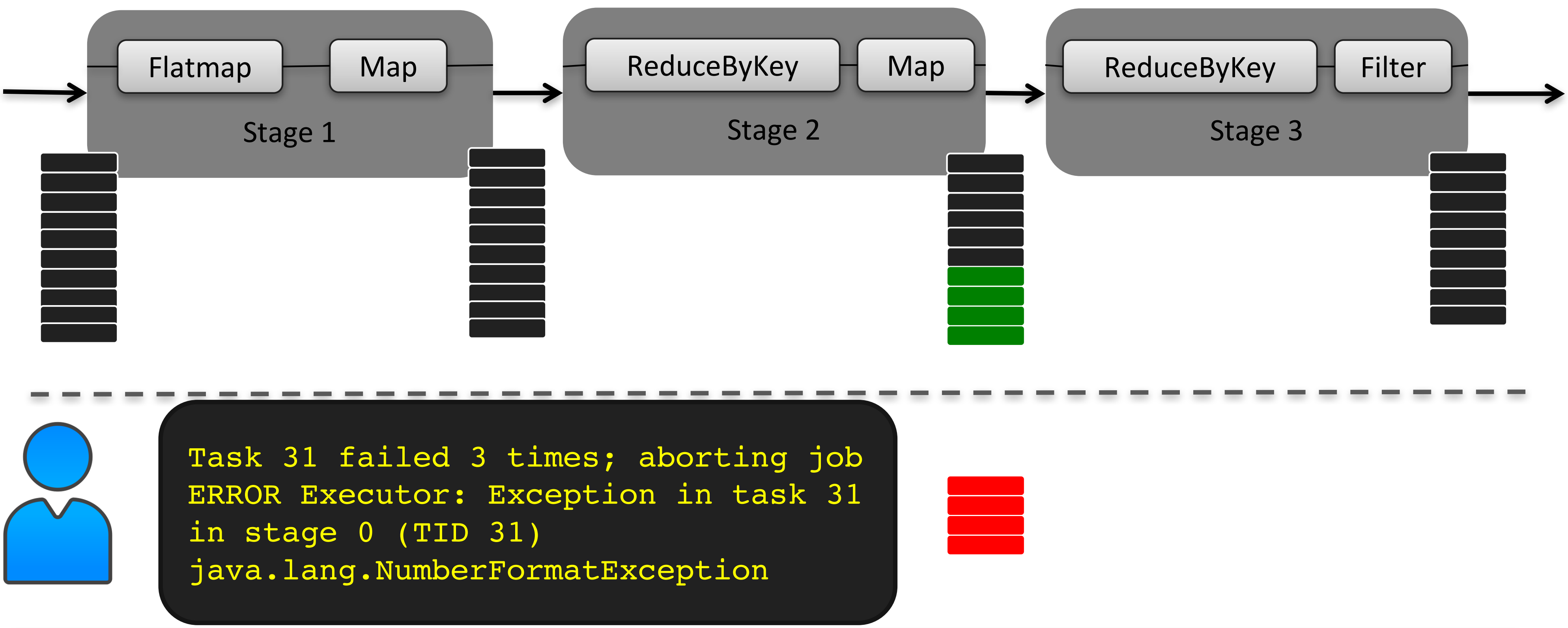
Allow a user to fix code after the breakpoint

2. On-Demand Guarded Watchpoint



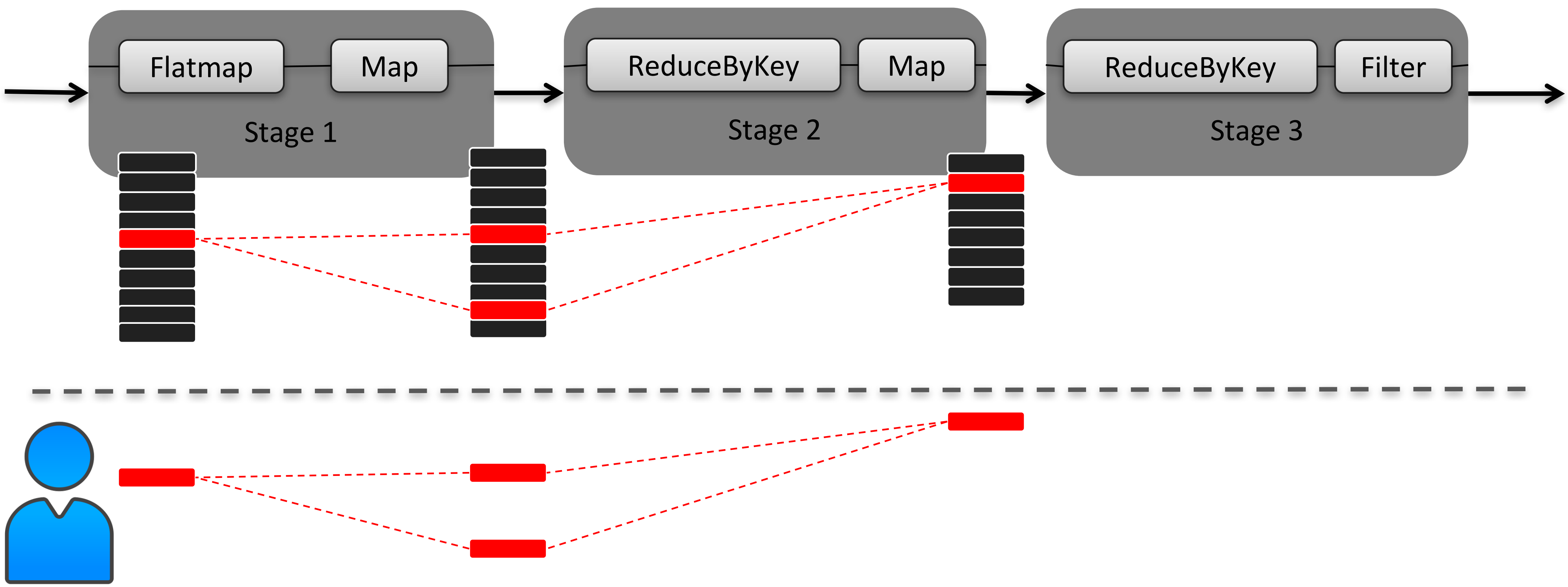
Watchpoint captures individual data records matching a user-provided guard

3. Crash Culprit Remediation



A user can either correct the crashed record, skip the crash culprit, or supply a code fix to repair the crash culprit.

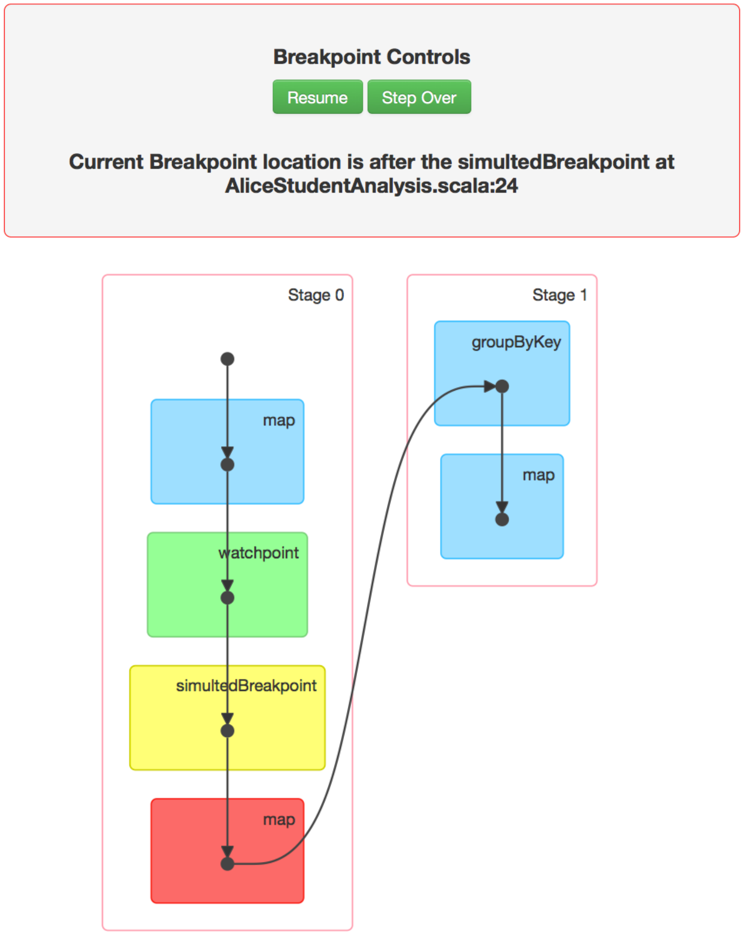
4. Backward and Forward Tracing



A user can also issue tracing queries on intermediate records at realtime

Demo: BigDebug Interactive Debugger

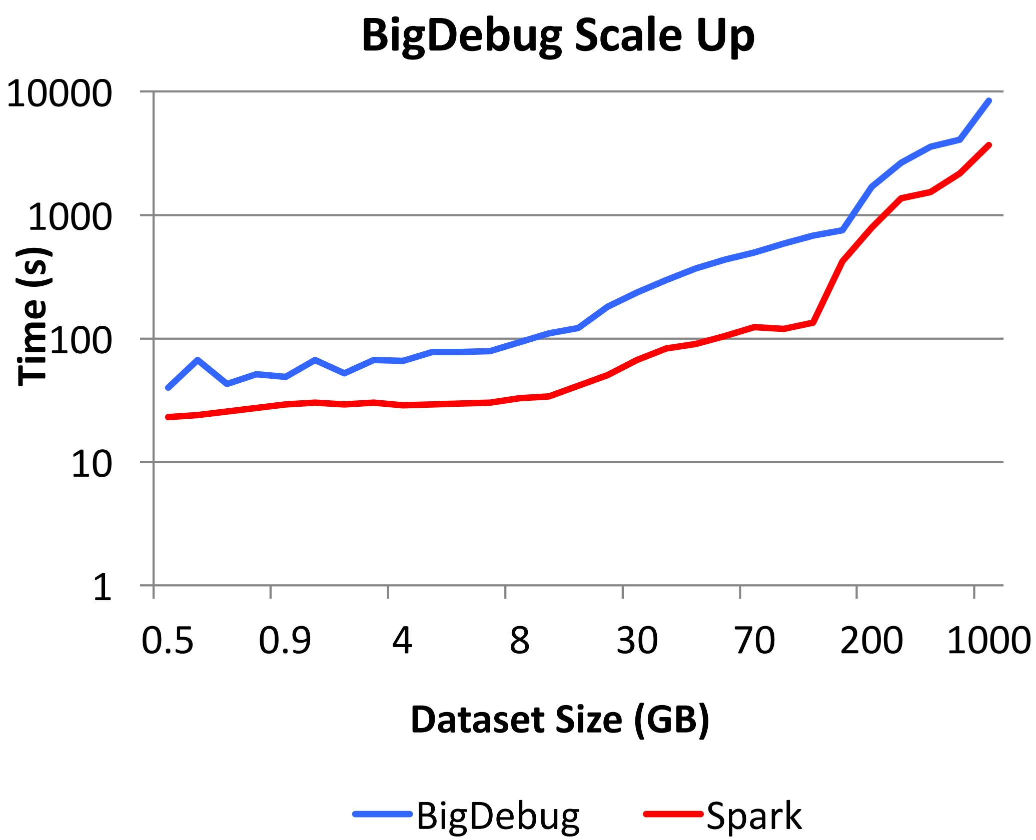
[FSE 2016 Demo, SIGMOD 2017 Demo]



AliceStudentAnalysis.scala

```scala
10 object AliceStudentAnalysis {
11
12   val COLLEGEYEAR = List("Sophomore" , "Freshman" , "Junior", "Senior")
13   def main(args: Array[String]): Unit = {
14
15     //set up spark configuration
16     val sparkConf = new SparkConf()
17     val bdconf = new BigDebugConfiguration
18     bdconf.setFilePath("/home/ali/work/temp/git/dsbigdebug/spark-lineage/exa
19     //set up spark context
20     val ctx = new SparkContext(sparkConf)
21     ctx.setBigDebugConfiguration(bdconf)
22     //spark program starts here
23     val records = ctx.textFile("/home/ali/Desktop/myfile.txt", 1).
24     simultedBreakpoint(s=> !COLLEGEYEAR.contains(s.split(" ")(2)))
25 >   val grade_age_pair = records.map(line => {
26       val list = line.split(" ")
27       (list(2), list(3).toInt)
28     })
29     val average_age_by_grade = grade_age_pair.groupByKey
30       .map(pair => {
31       val itr = pair._2.toIterator
32       var moving_average = 0
33       var num = 1
34       while (itr.hasNext) {
35         moving_average = moving_average + itr.next()
36         num = num + 1
37       }
38       (pair._1, moving_average/num)
39     })
40     val out = average_age_by_grade.collect()
41     out.foreach(println)
42   }
43 }
44
```

Q1 : How does BigDebug scale to massive data?



BigDebug retains scale up property of Spark. This property is critical for Big Data processing frameworks

Q2 : What is the performance overhead of debugging primitives?

Program	Dataset size (GB)	Max	Max w/o Latency Alert	Watchpoint	Crash Culprit	Tracing
WordCount	0.5 - 1000	2.5X	1.34X	1.09X	1.18X	1.22X
Grep	20 - 90	1.76X	1.07X	1.05X	1.04X	1.05X
PigMix-L1	1 - 200	1.38X	1.29X	1.03X	1.19X	1.24X

Max : All the features of BigDebug are enabled

BigDebug poses at most 2.5X overhead with the maximum instrumentation setting.

Data Scientists | Big Data Debugging

Titian: Data Provenance Support in Spark

Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, Tyson Condie
[42nd Conference on Very Large Data Bases, VLDB 2016]

UCLA

Data Provenance – Example in SQL

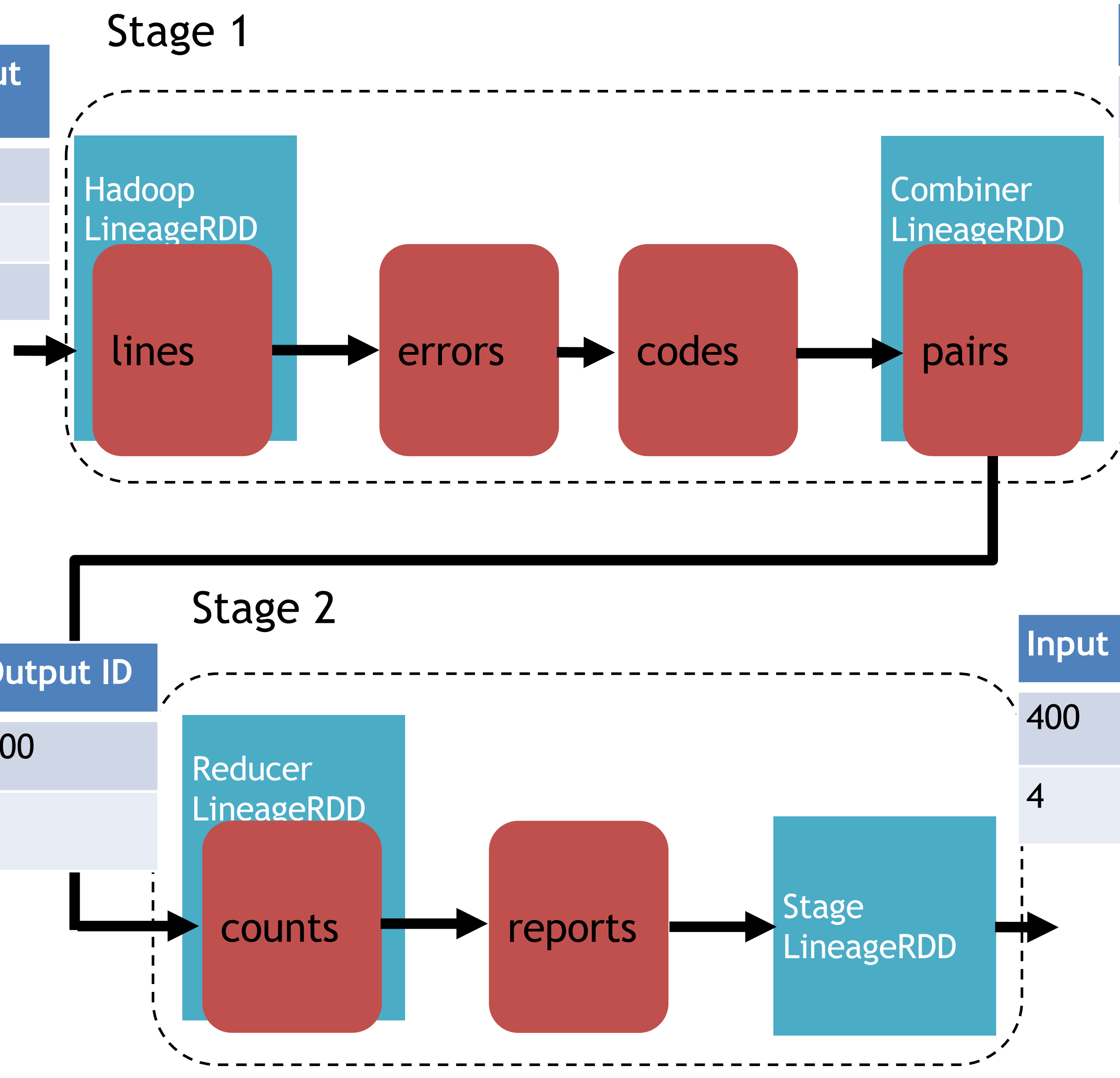
```
SELECT time, AVG(temp)
  FROM sensors
  GROUP BY time
```

Sensors			
Tuple-ID	Time	Sendor-ID	Temperatur e
T1	11AM	1	34
T2	11AM	2	35
T3	11AM	3	35
T4	12PM	1	35
T5	12PM	2	35
T6	12PM	3	100
T7	1PM	1	35
T8	1PM	2	35
T9	1PM	[illegible]	80

Result-ID	Time	AVG(temp)	
ID-1	11AM	34.6	
ID-2	12PM	56.6	Outlier
ID-3	1PM	50	Outlier

Why ID-2 and ID-3 have those high values?

Step 1: Instrumented Workflow in Spark



Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
[p1, p2]	400
[p1]	4

Input ID	Output ID
400	id1
4	id2

Step 2: Example Backward Tracing

Hadoop	
Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Combiner	
Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
p1	400

Reducer	
Input ID	Output ID
[p1, p2]	400
[p1]	4

Stage	
Input ID	Output ID
400	id1
4	id2

Hadoop	
Input ID	Output ID
offset1	id1
…	…

Combiner	
Input ID	Output ID
{ id1, …}	400

Input ID	Output ID
p1	400

Reducer.Output ID $\bowtie$ Stage.Input ID

Step 2: Example Backward Tracing

Hadoop	
Input ID	**Output ID**
offset1	id1
offset2	id2
offset3	id3

Combiner	
Input ID	**Output ID**
{ id1, id 3}	400
{ id2 }	4

Input ID	Output ID
p1	400

Combiner.Output ID ⋈ Reducer.Output ID

Hadoop	
Input ID	**Output ID**
offset1	id1
…	…

Combiner	
Input ID	**Output ID**
{ id1, …}	400

Input ID	Output ID
p1	400

Combiner.Output ID ⋈ Reducer.Output ID

Step 2: Example Backward Tracing

Hadoop	
Input ID	Output ID
offset1	id1
offset2	id2
offset3	id3

Combiner	
Input ID	Output ID
{ id1, id 3}	400
{ id2 }	4

Hadoop.Output ID $\bowtie$ Combiner.Input ID

Hadoop	
Input ID	Output ID
offset1	id1
...	...

Combiner	
Input ID	Output ID
{ id1, ...}	400

Hadoop.Output ID $\bowtie$ Combiner.Input ID

Data Scientists
Big Data Debugging

Automated Debugging in Data Intensive Scalable Computing

Muhammad Ali Gulzar, Matteo Interlandi, Xueyuan Han, Mingda Li
Tyson Condie, Miryung Kim

[ACM Symposium on Cloud Computing, SoCC 2017]

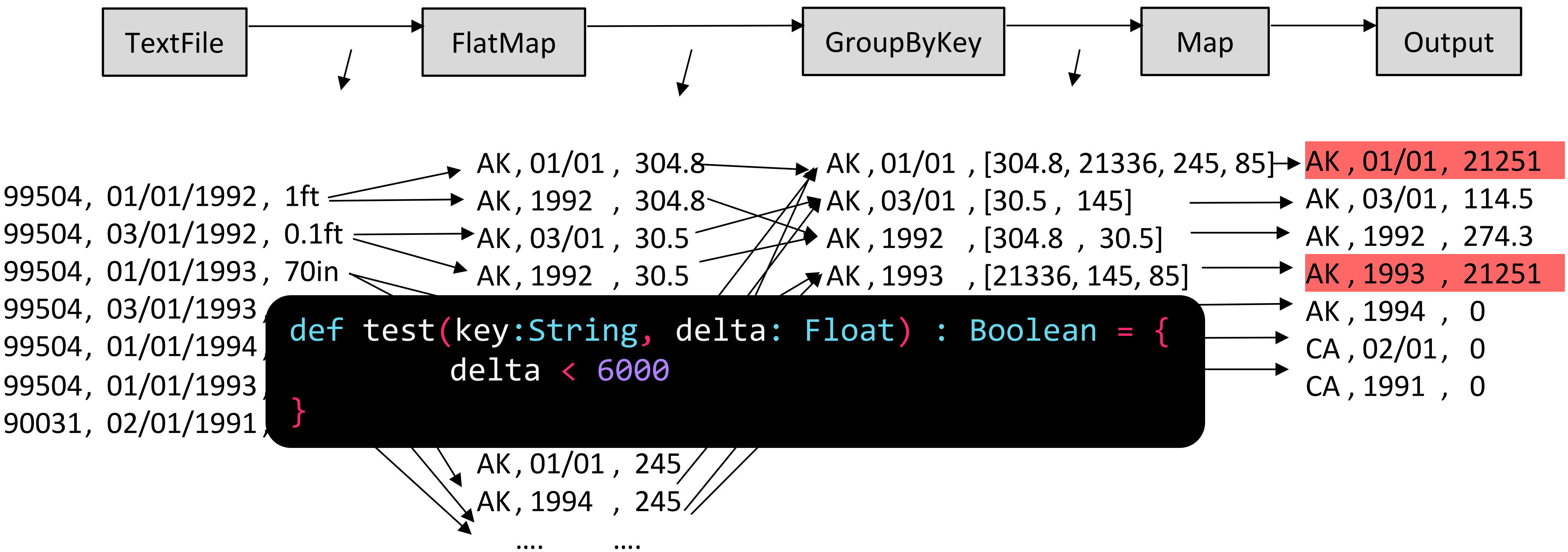
Motivating Example

- 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**.
- An input data record that measures 1 foot of snowfall on January 1st of Year 1992, in the 99504 zip code (Anchorage, AK) area, appears as

99504 , 01/01/1992 , 1ft

Problem Definition

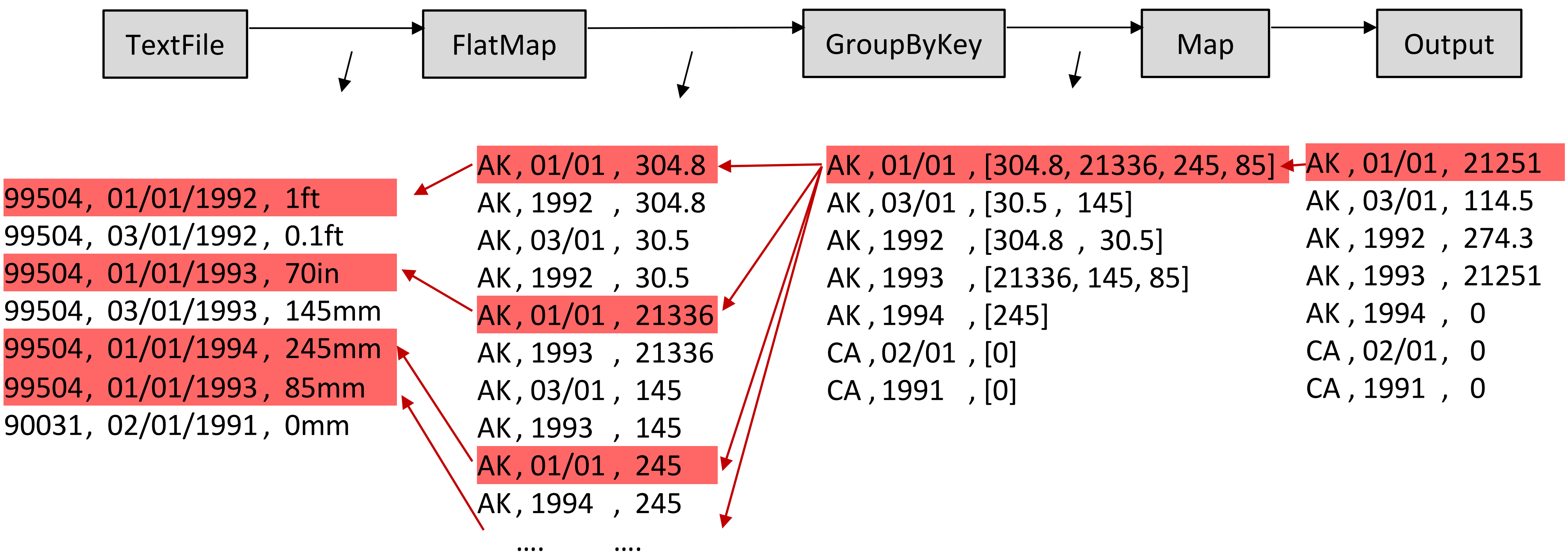
- Using a test function, a user can specify incorrect results




```
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.

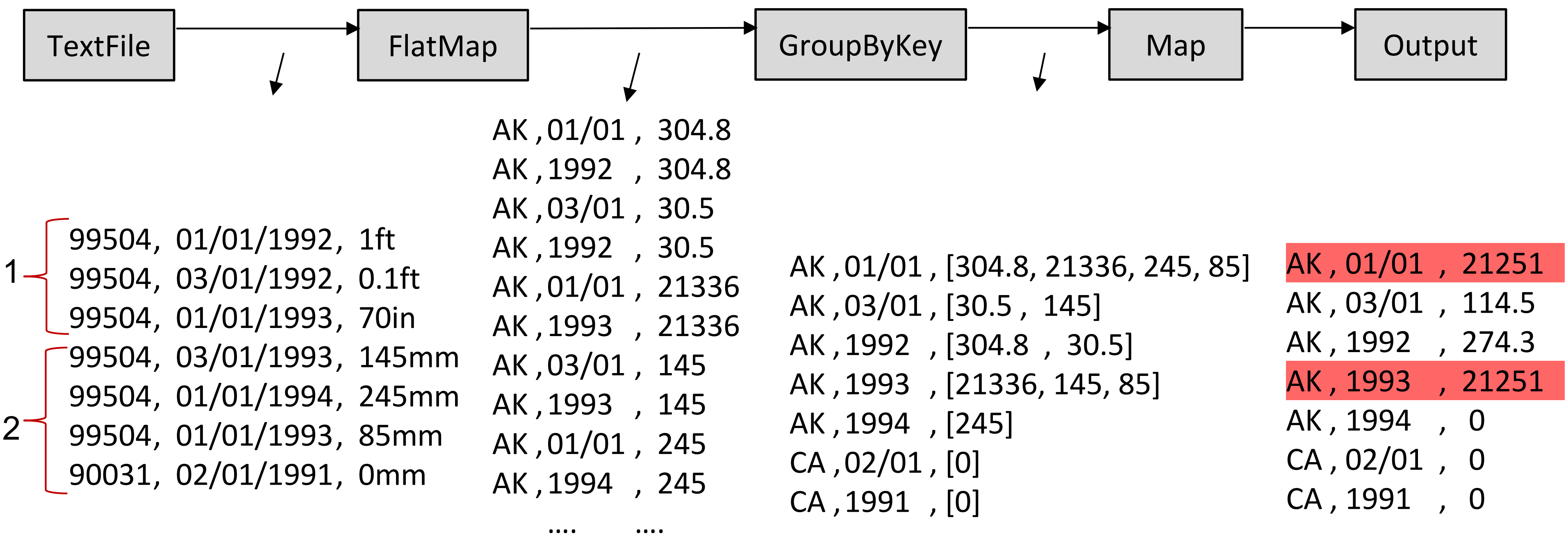
Existing Approach 1: Data Provenance for Spark



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



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 search-like procedure on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

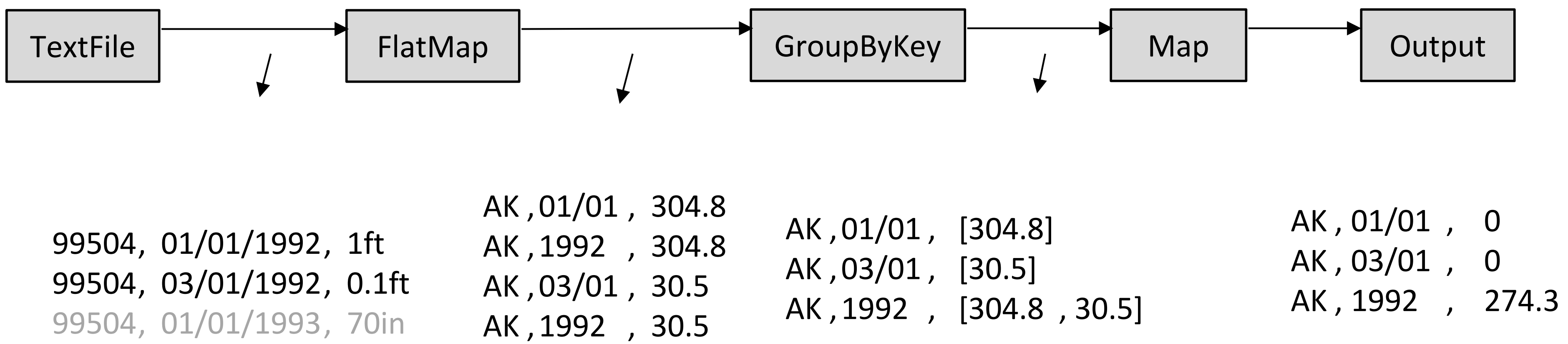
1 { 99504, 01/01/1992, 1ft 99504, 03/01/1992, 0.1ft 2 { 99504, 01/01/1993, 70in	AK , 01/01 , 304.8 AK , 1992 , 304.8 AK , 03/01 , 30.5 AK , 1992 , 30.5 AK , 01/01 , 21336 AK , 1993 , 21336	AK , 01/01 , [304.8, 21336] AK , 03/01 , [30.5] AK , 1992 , [304.8 , 30.5] AK , 1993 , [21336]	AK , 01/01 , 21031 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 search-like procedure on the input dataset using a test oracle function

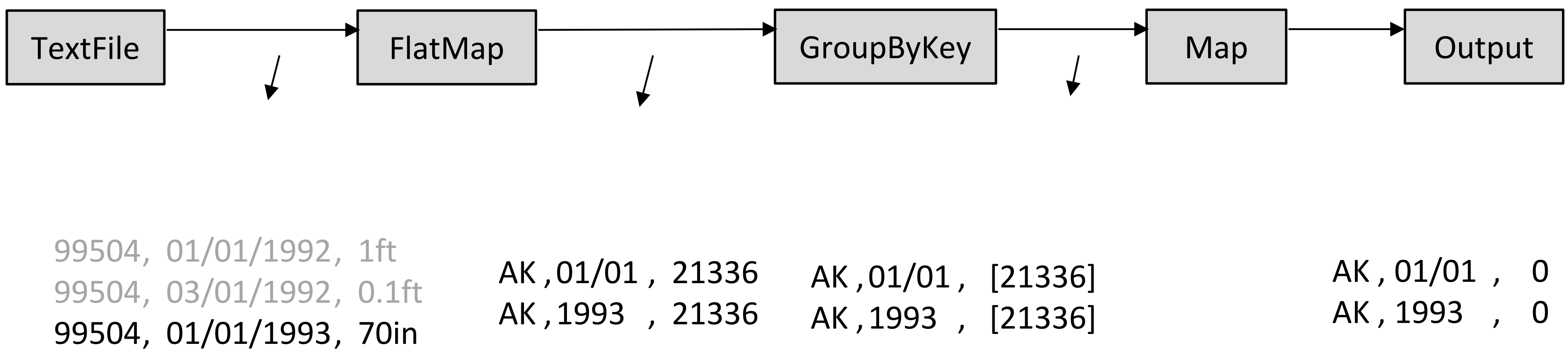


Run 3

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 search-like procedure on the input dataset using a test oracle function

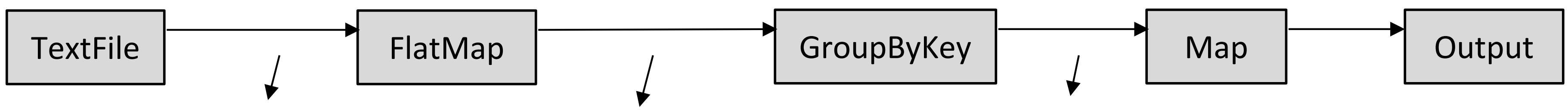


Run 4

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 search-like procedure on the input dataset using a test oracle function



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 , [304.8]
AK , 1992 , [304.8]

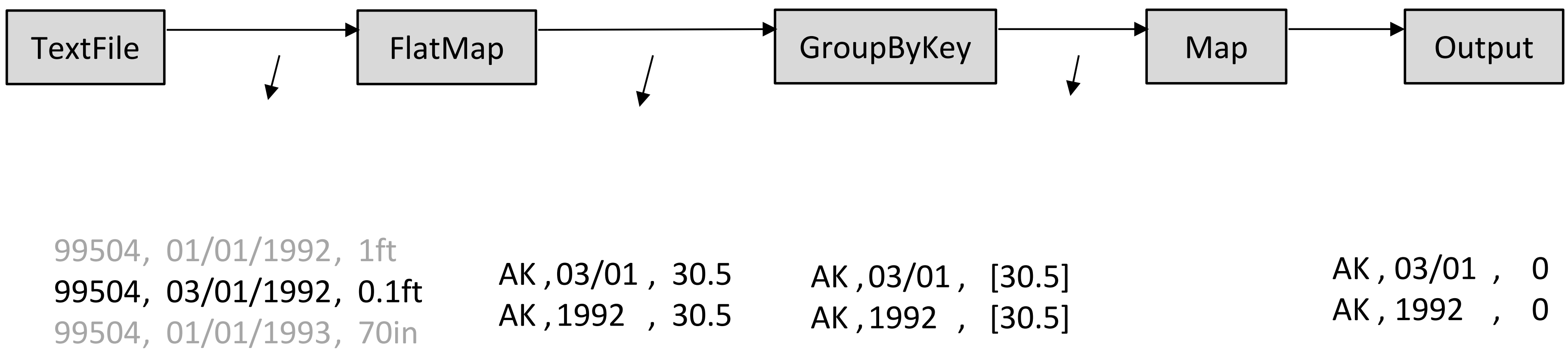
AK , 01/01 , 0
AK , 1992 , 0

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 search-like procedure 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 search-like procedure on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

TextFile → FlatMap	FlatMap → GroupByKey	GroupByKey → Map	Output
99504, 01/01/1992, 1ft	AK ,01/01 , 21336	AK ,01/01 , [21336]	AK , 01/01 , 0
99504, 03/01/1992, 0.1ft	AK ,1993 , 21336	AK ,1993 , [21336]	AK , 1993 , 0
99504, 01/01/1993, 70in			

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 search-like procedure on the input dataset using a test oracle function

TextFile → FlatMap → GroupByKey → Map → Output

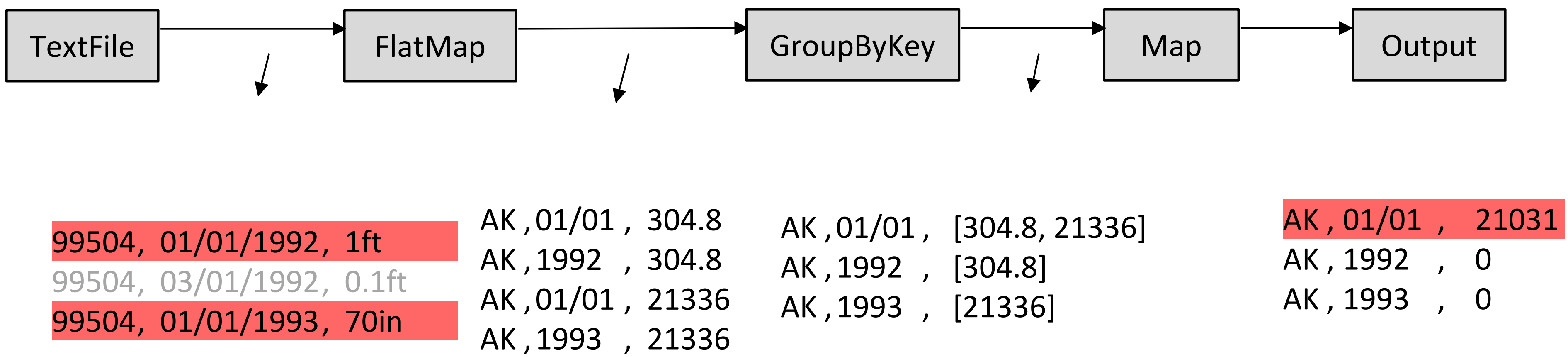
99504, 01/01/1992, 1ft	AK , 03/01 , 30.5	AK , 01/01 , [21336]	AK , 01/01 , 0
99504, 03/01/1992, 0.1ft	AK , 1992 , 30.5	AK , 03/01 , [30.5]	AK , 03/01 , 0
99504, 01/01/1993, 70in	AK , 01/01 , 21336	AK , 1992 , [30.5]	AK , 1992 , 0
	AK , 1993 , 21336	AK , 1993 , [21336]	AK , 1993 , 0

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

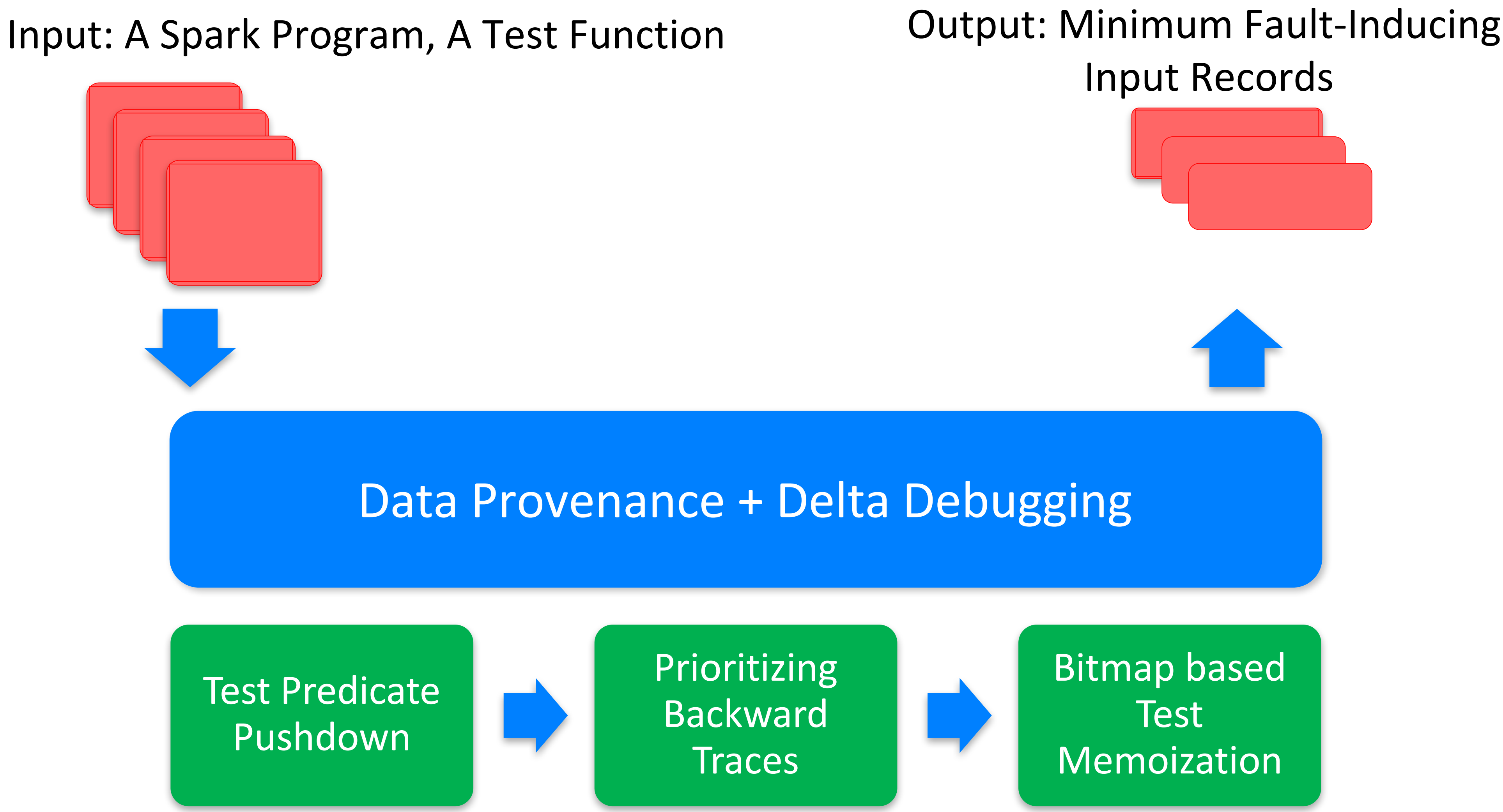
- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function



Run 9

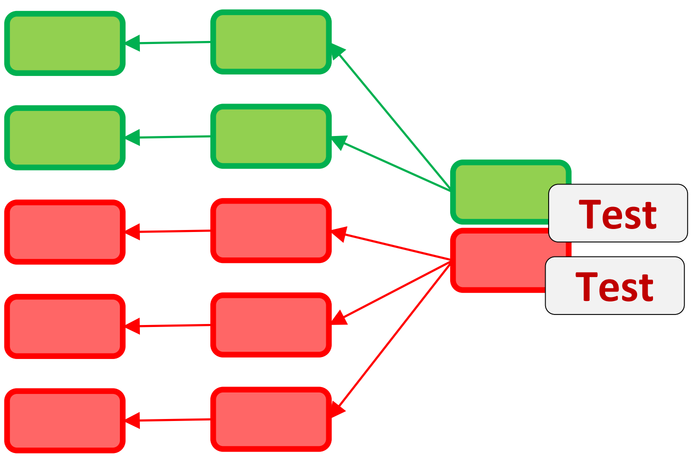
It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

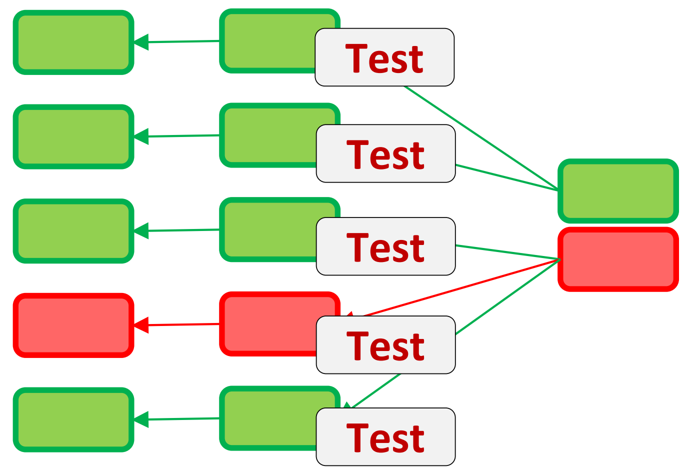
Automated Debugging in DISC with BigSift



Optimization 1: Test Predicate Pushdown

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

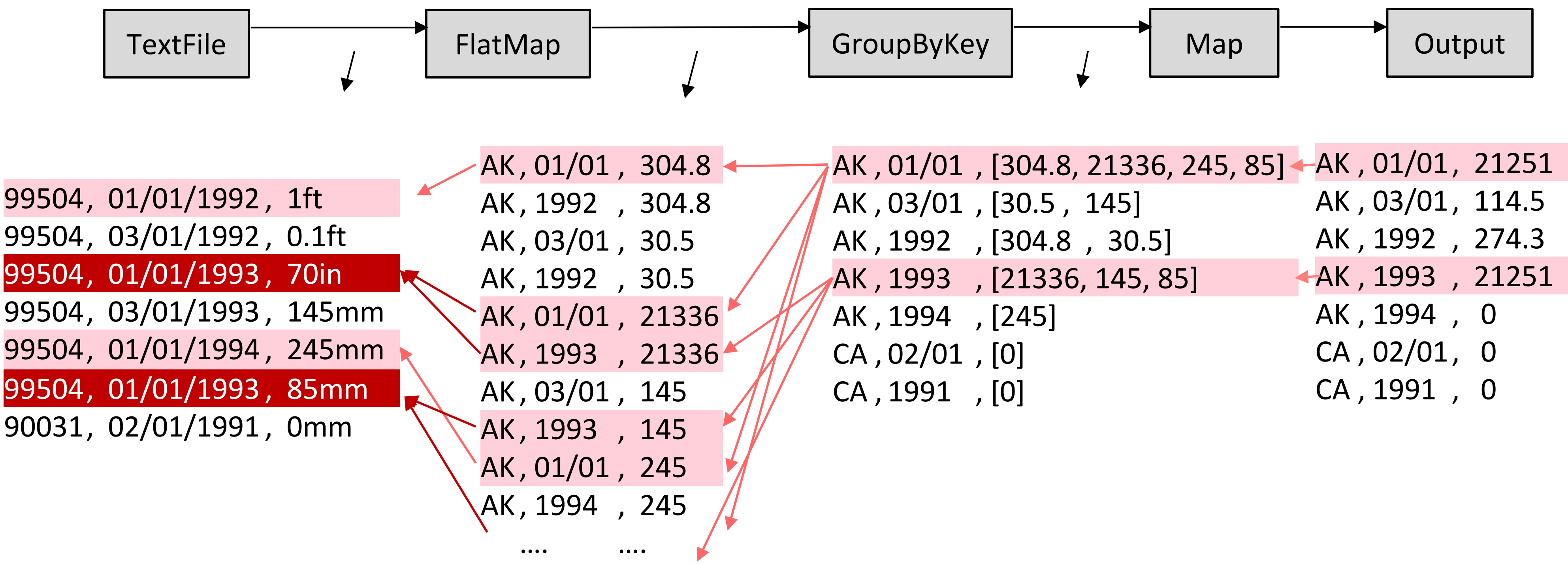




If applicable, BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.

Optimization 2: Prioritizing Backward Traces

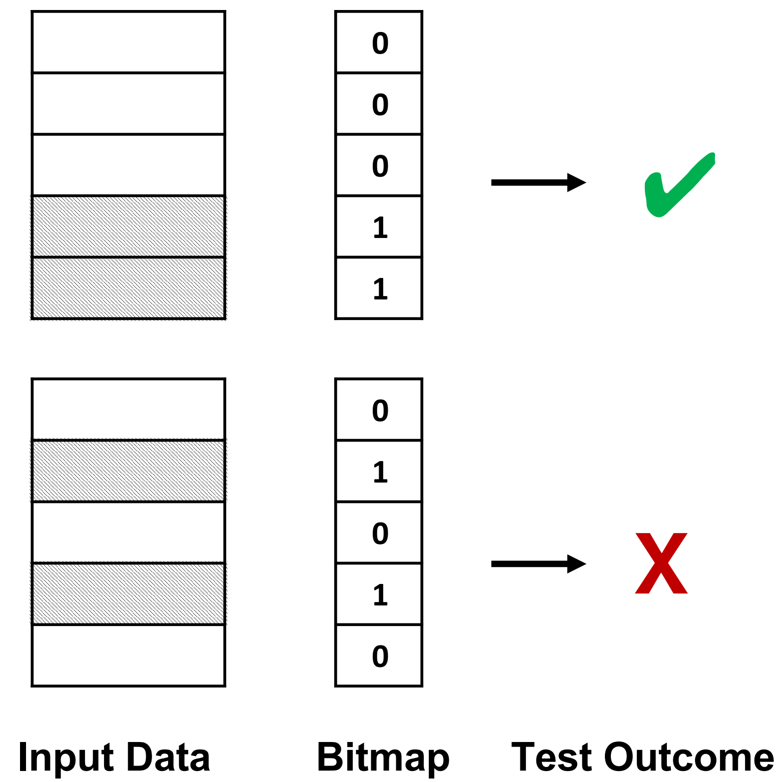
- **Observation:** The same faulty input record may contribute to multiple output records failing the test.



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.

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

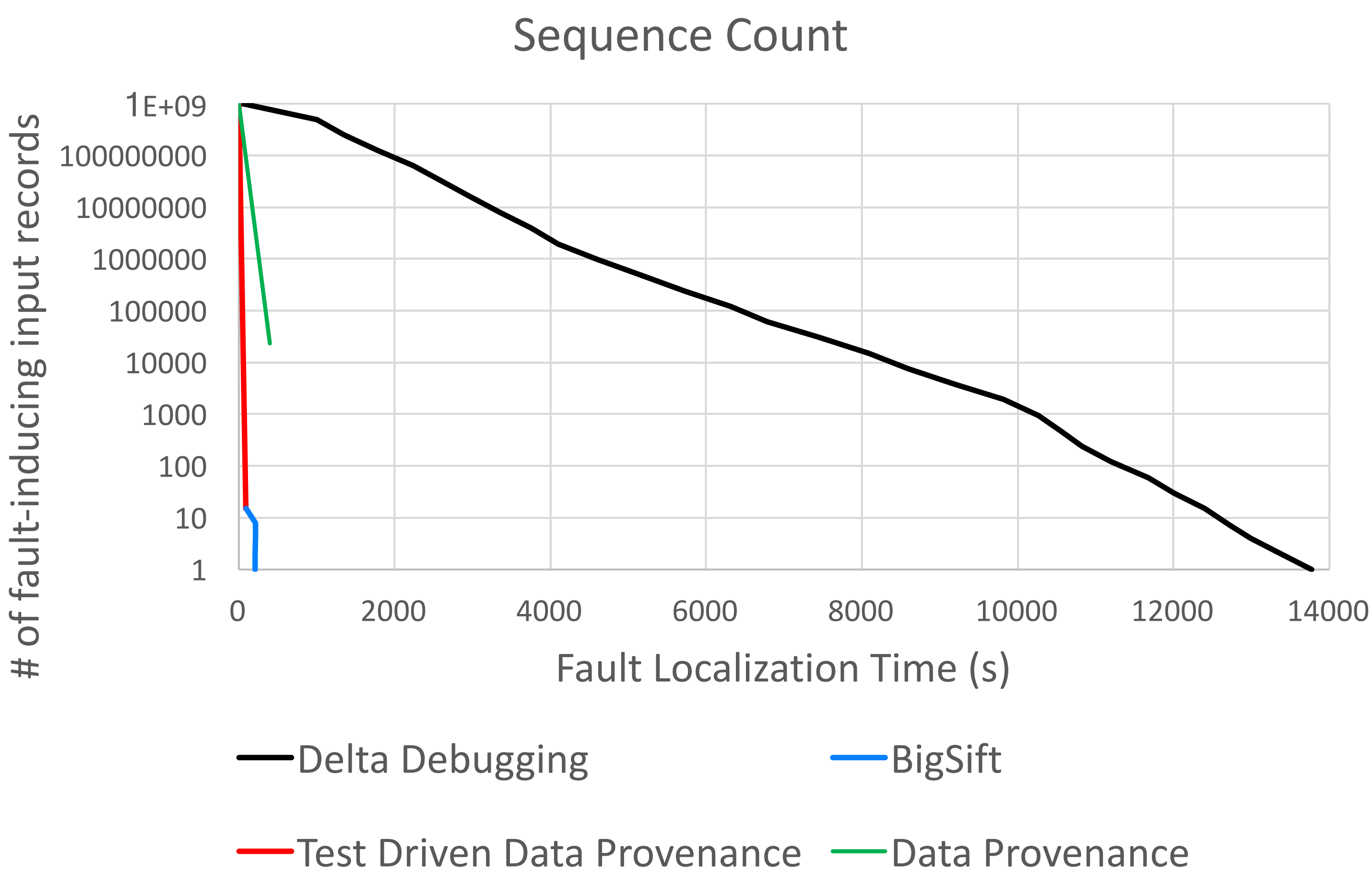
BigSift provides up to a 66X speed up in isolating the precise fault-inducing input records, in comparison to the baseline DD

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

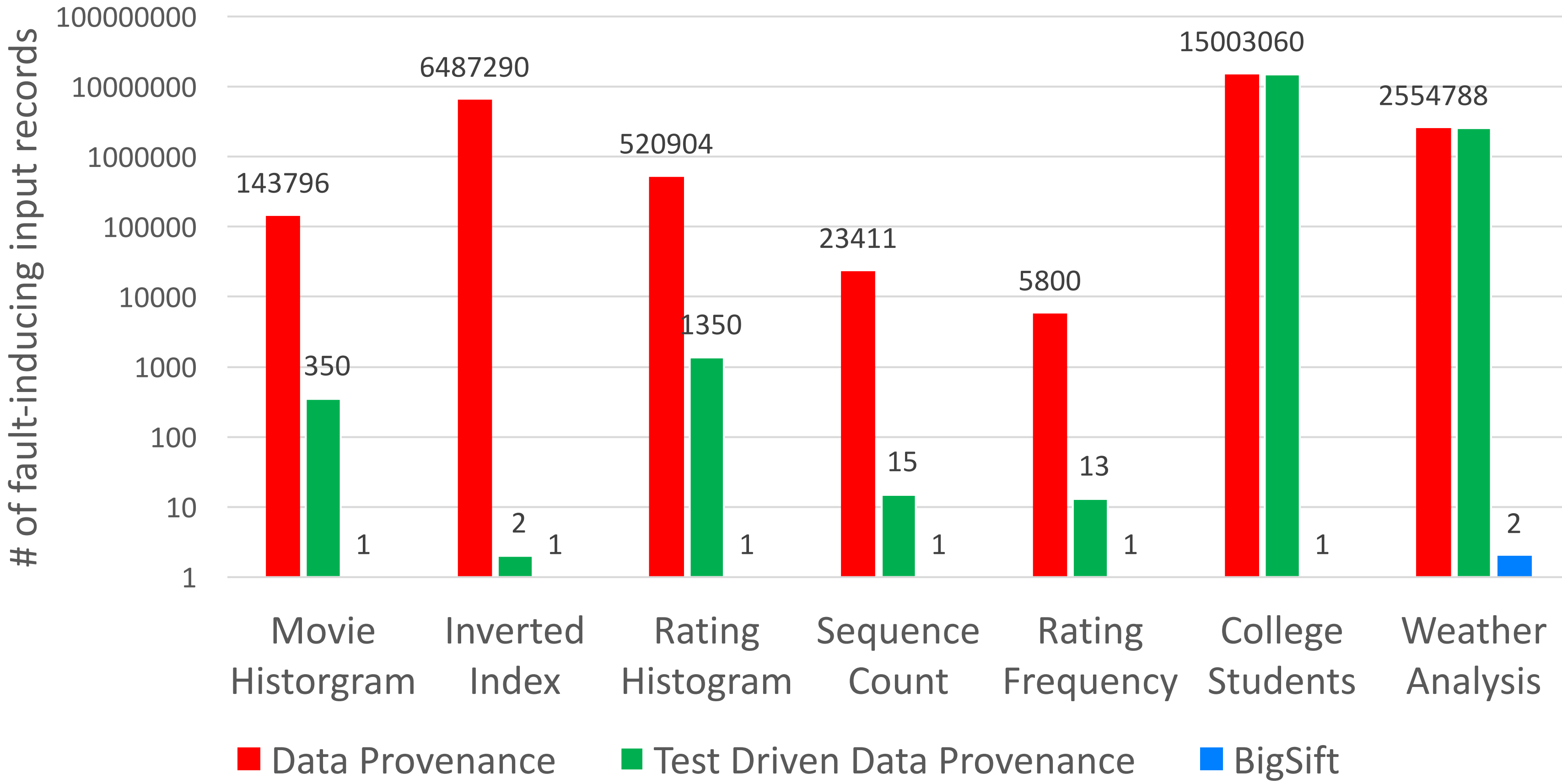
On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

RQ2: Debugging Time



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

Software Engineering *elevating* Data Science

Data Scientists in Software Teams
[ICSE ‘16, TSE ’18]

- Background
- Work Activities
- Challenges
- Best Practices
- Quality Assurance

Debugging for Big Data Analytics

- Interactive Debugger [ICSE ’16]
- Data Provenance [VLDB ‘16]
- Automated Debugging [SoCC ‘17]

Optimization for Iterative Development

- “How can we re-compute big data analytics in case of code changes?” [SoCC ‘16]

Automated Testing for Big Data Analytics

- “How do we help select (sample) data for local testing?”
- “How do we generate test data to achieve high code coverage?”

Data Summary and Explanation

- “How we do characterize data by inferring the underlying type and format?”

Late Stage Customization of Big Data System Stack

- “How do we customize Big Data runtime for the actual use of big data analytics?”

Big Data needs **awesome** software engineering **tools**

Diagnose	Fix	Optimize
		
✓ Debugging ✓ Intelligent sampling and testing ✓ Root cause analysis	✓ Data cleaning	✓ Performance analytics ✓ Code analytics