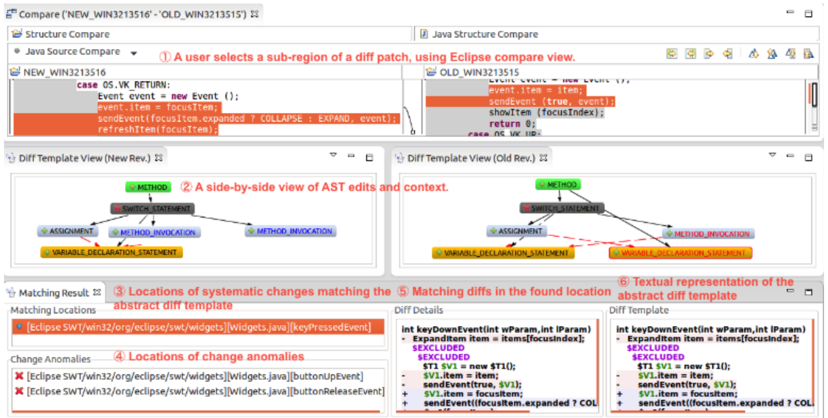
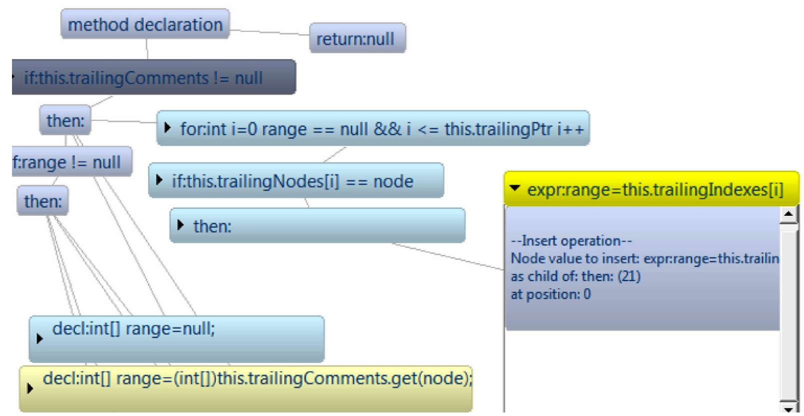
Data Science *elevating* Software Engineering
Software Engineering *elevating* Data Science

Miryung Kim
University of California, Los Angeles

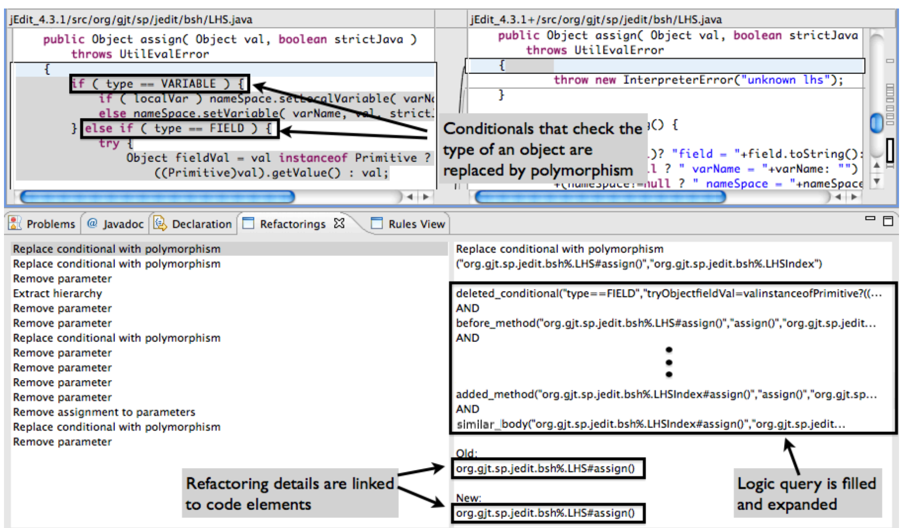
Software Engineering and Analysis Lab at UCLA



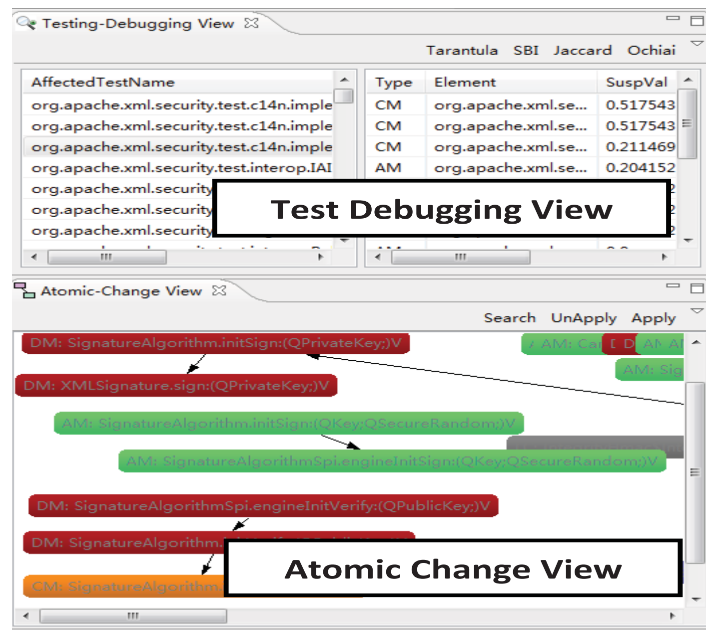
Interactive Code Review / Clone Search



Refactoring and Transformation



Program Comprehension



Debugging

Data Science *elevating* Software Engineering

Empirical Studies of Software Changes

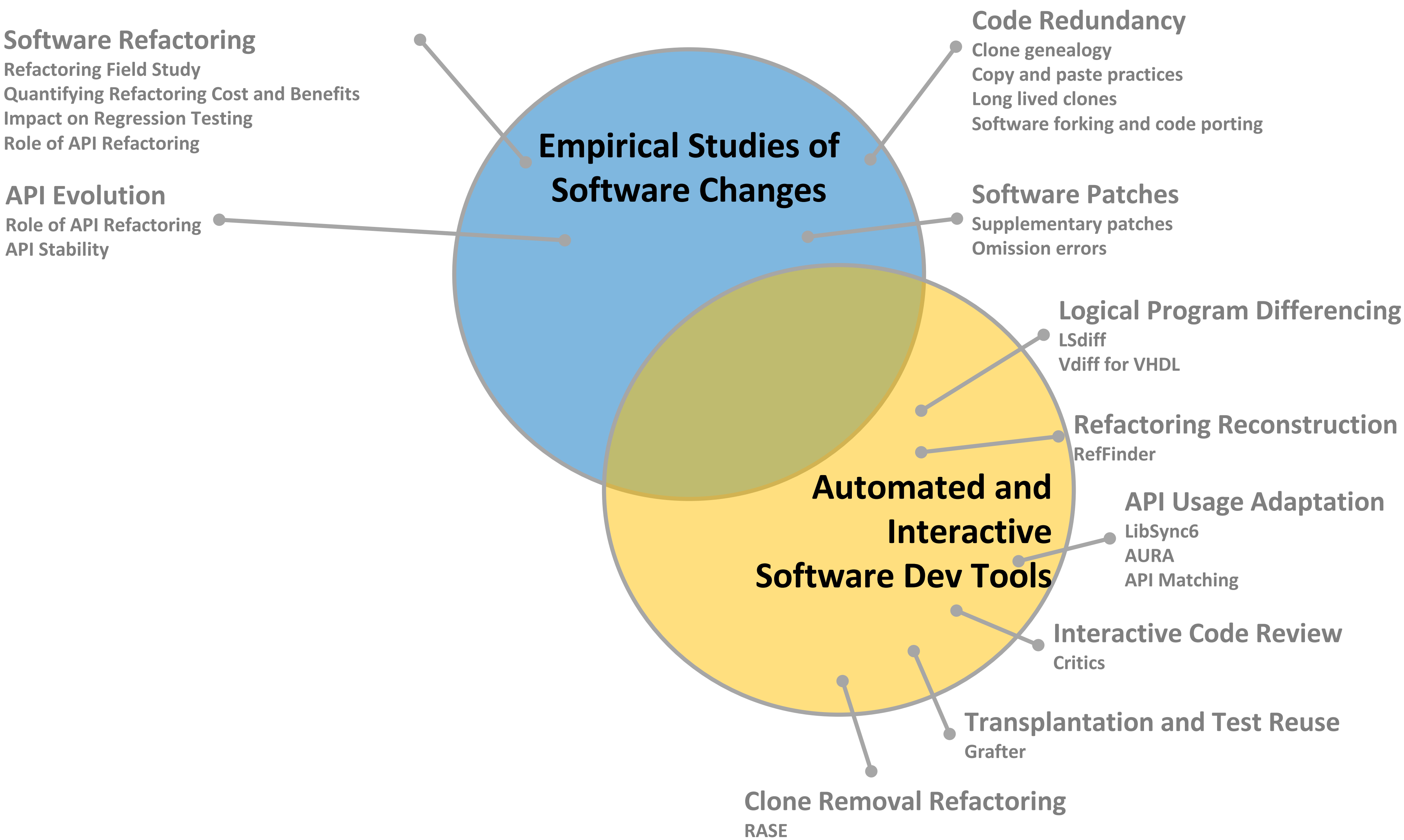
Software Refactoring
- Refactoring Field Study
- Quantifying Refactoring Cost and Benefits
- Impact on Regression Testing
- Role of API Refactoring

API Evolution
- Role of API Refactoring
- API Stability

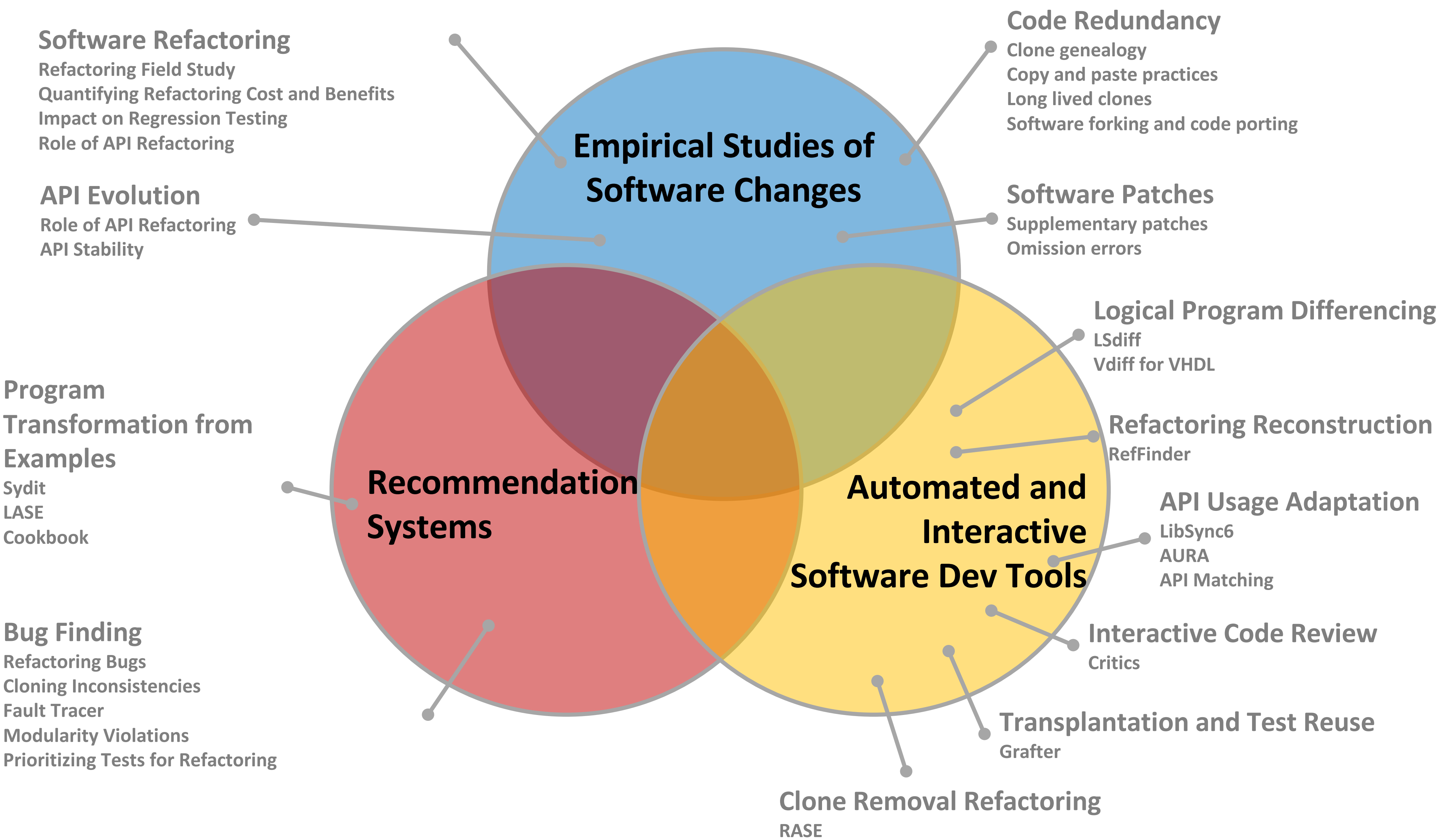
Code Redundancy
- Clone genealogy
- Copy and paste practices
- Long lived clones
- Software forking and code porting

Software Patches
- Supplementary patches
- Omission errors

Data Science *elevating* Software Engineering



Data Science *elevating* Software Engineering



Part 1:
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

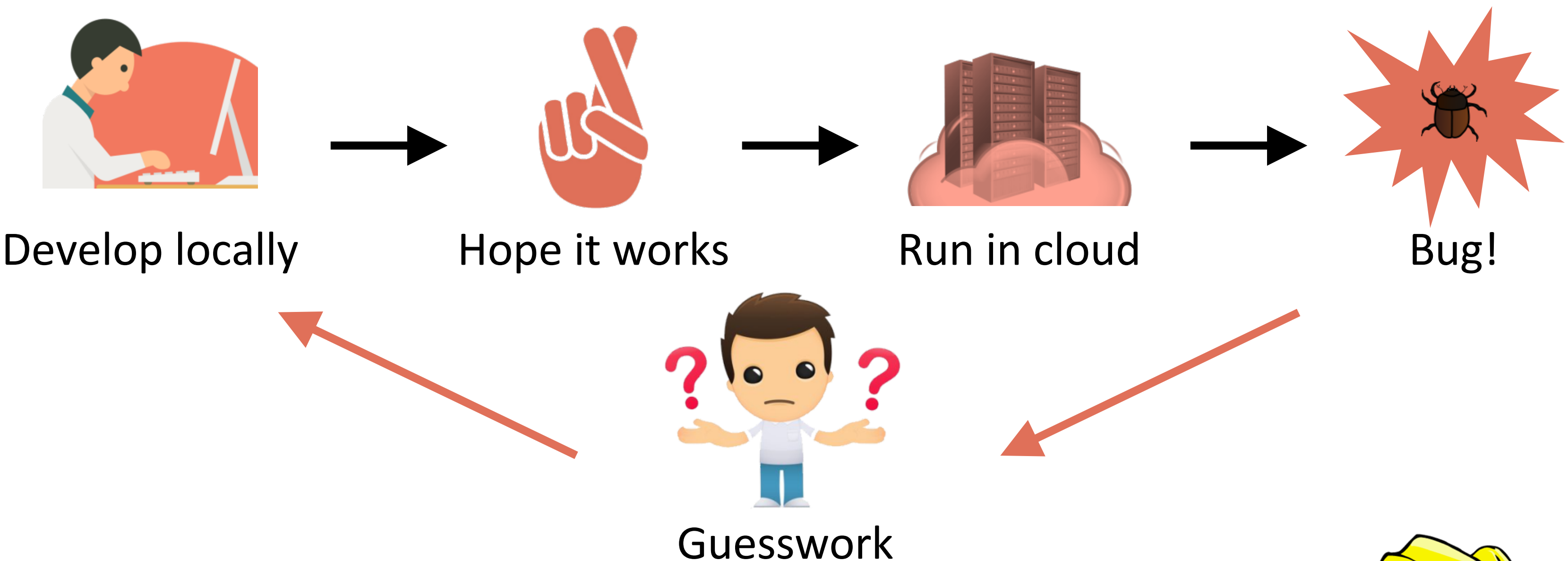
Challenges in Ensuring "Correctness"

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

Data Scientists | Big Data Debugging

BigDebug: Debugging Primitives for Interactive Big Data Processing in Spark

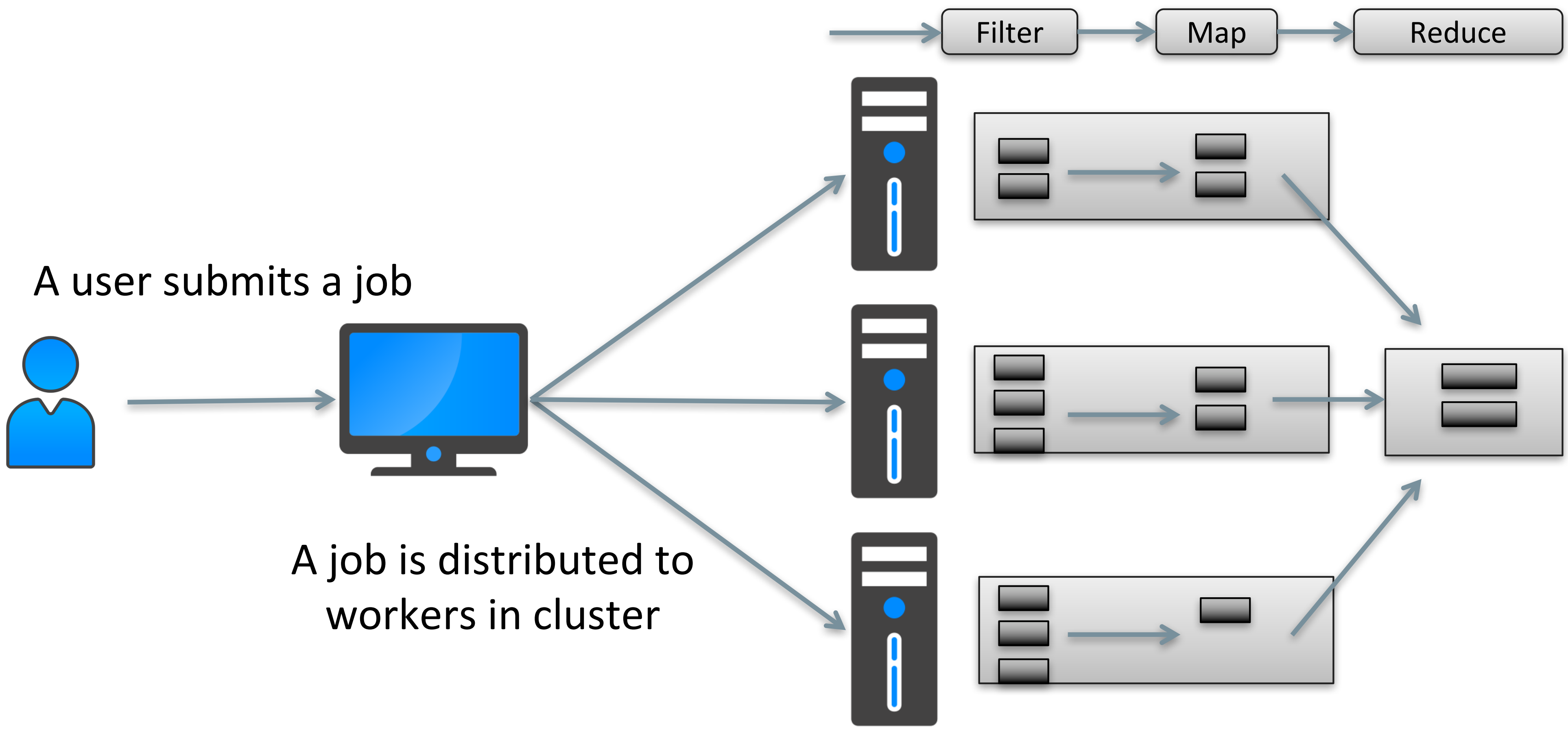
Muhammad Ali Gulzar, Matteo Interlandi, Seunghyun Yoo, Sai Deep Tetali, Tyson Condie, Todd Millstein, Miryung Kim

[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()
```

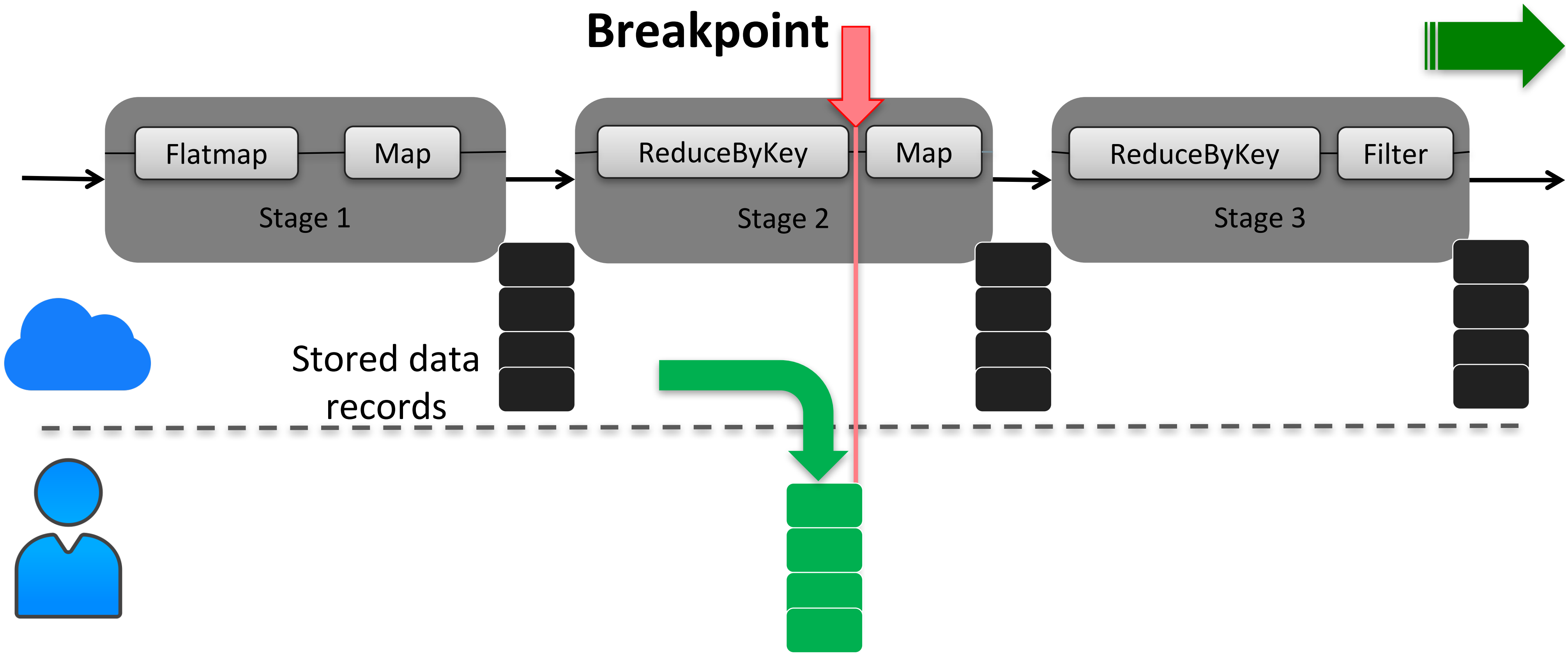
```
Task 31 failed 3 times; aborting
job
ERROR Executor: Exception in
task 31 in stage 0 (TID 31)

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

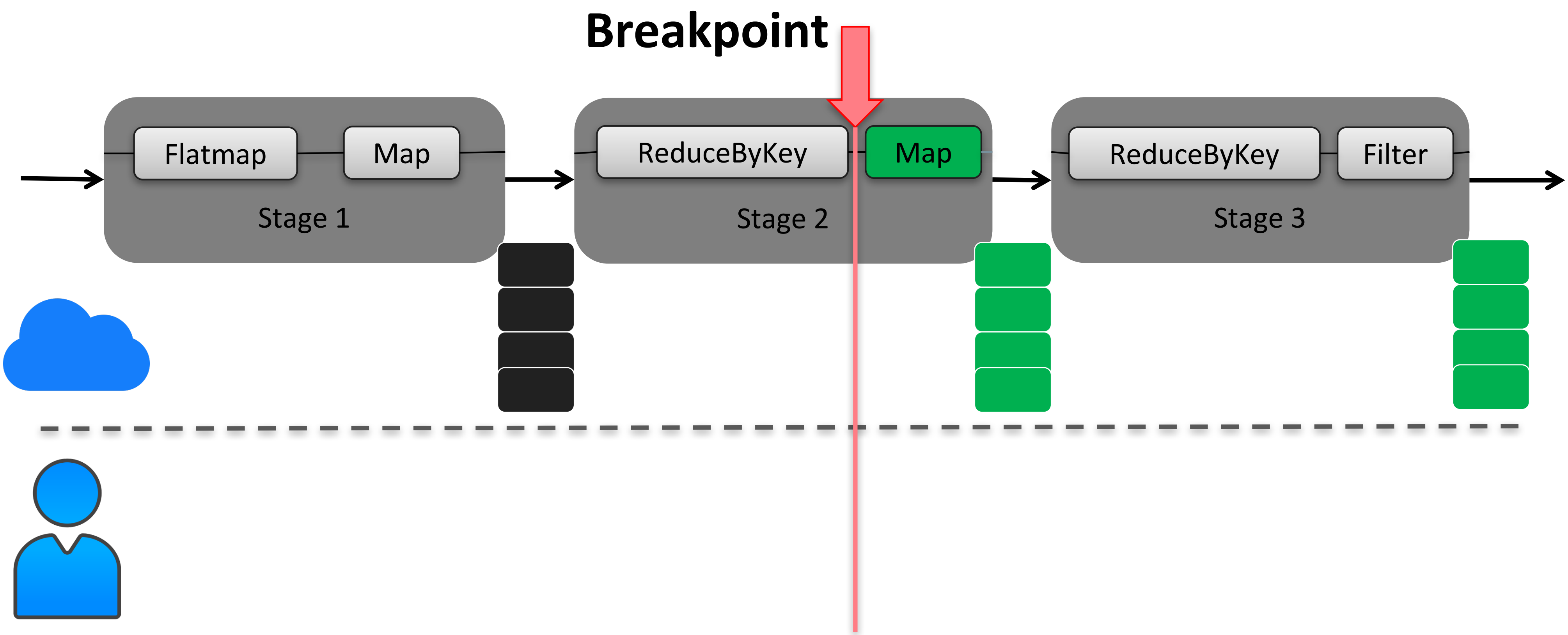
- Pausing the entire computation on the cloud could reduce throughput
- It is clearly infeasible for a user to inspect billion of records through a regular watchpoint
- Even launching remote JVM debuggers to individual worker nodes cannot scale for big data computing

1. Simulated Breakpoint



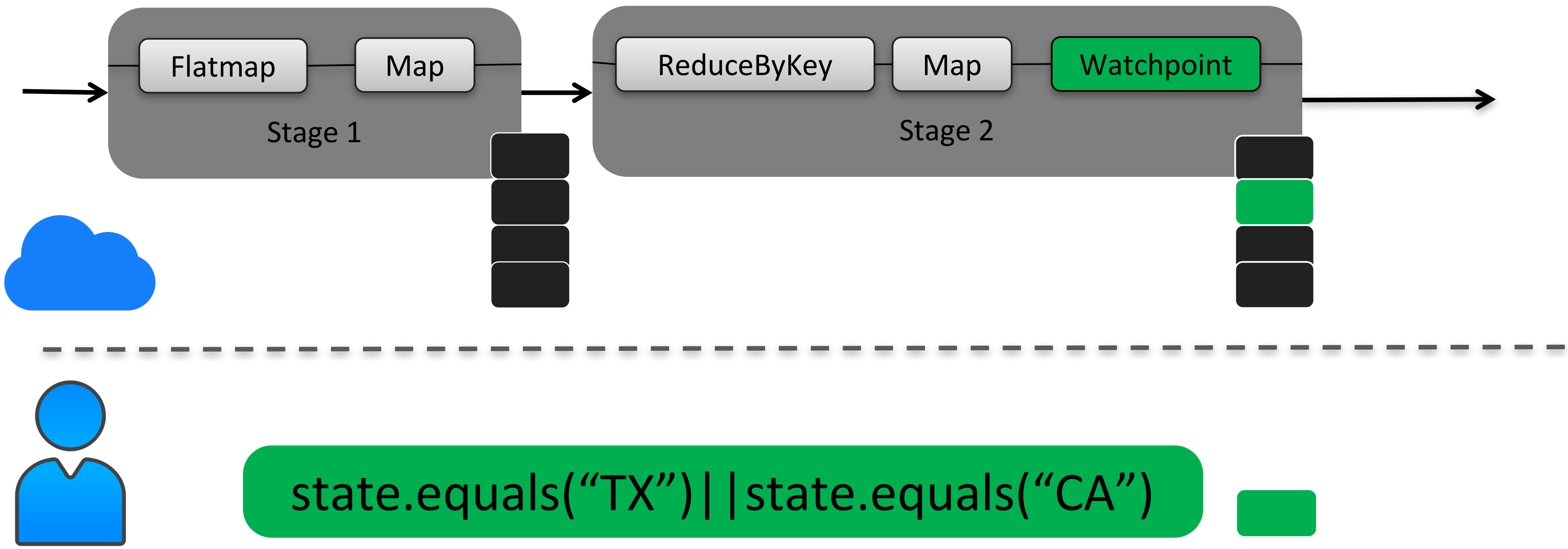
Simulated breakpoint replays computation from the latest materialization point where data is stored in memory

1. Simulated Breakpoint – Realtime Code Fix



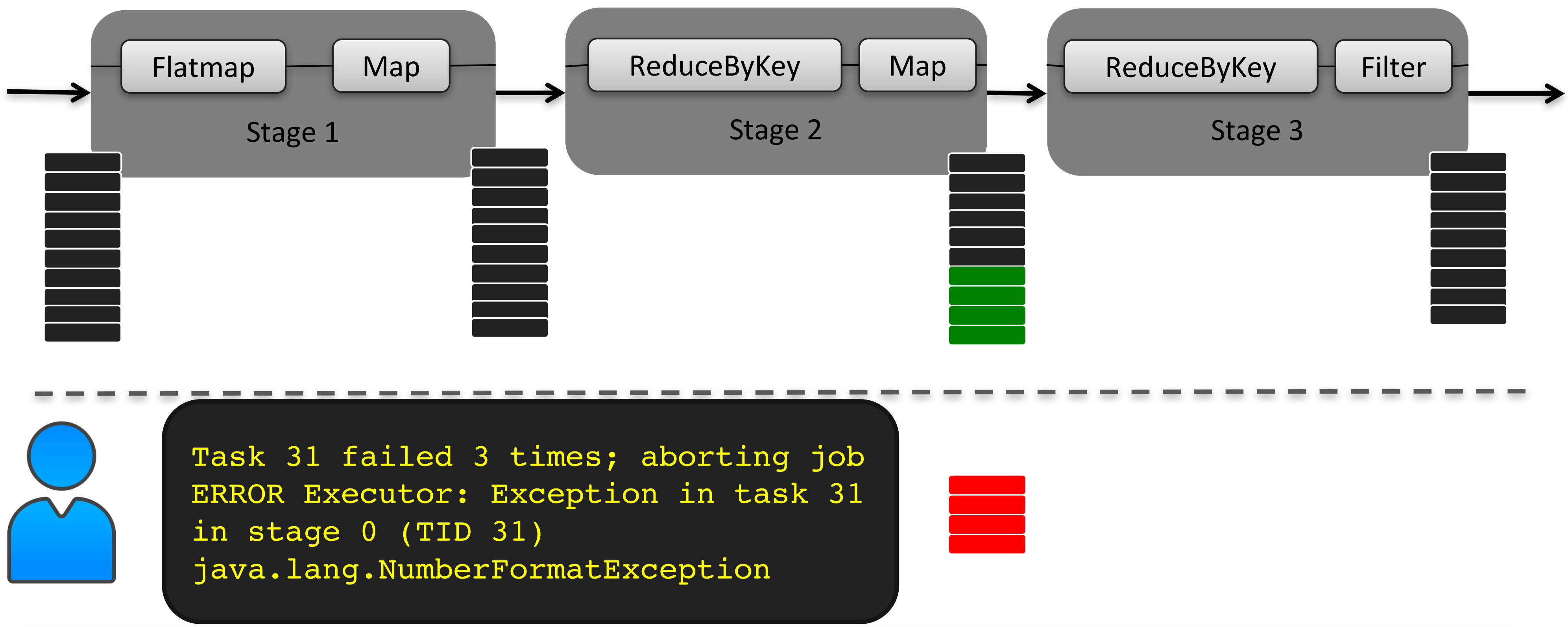
Allow a user to fix code after the breakpoint

2. On-Demand Guarded Watchpoint



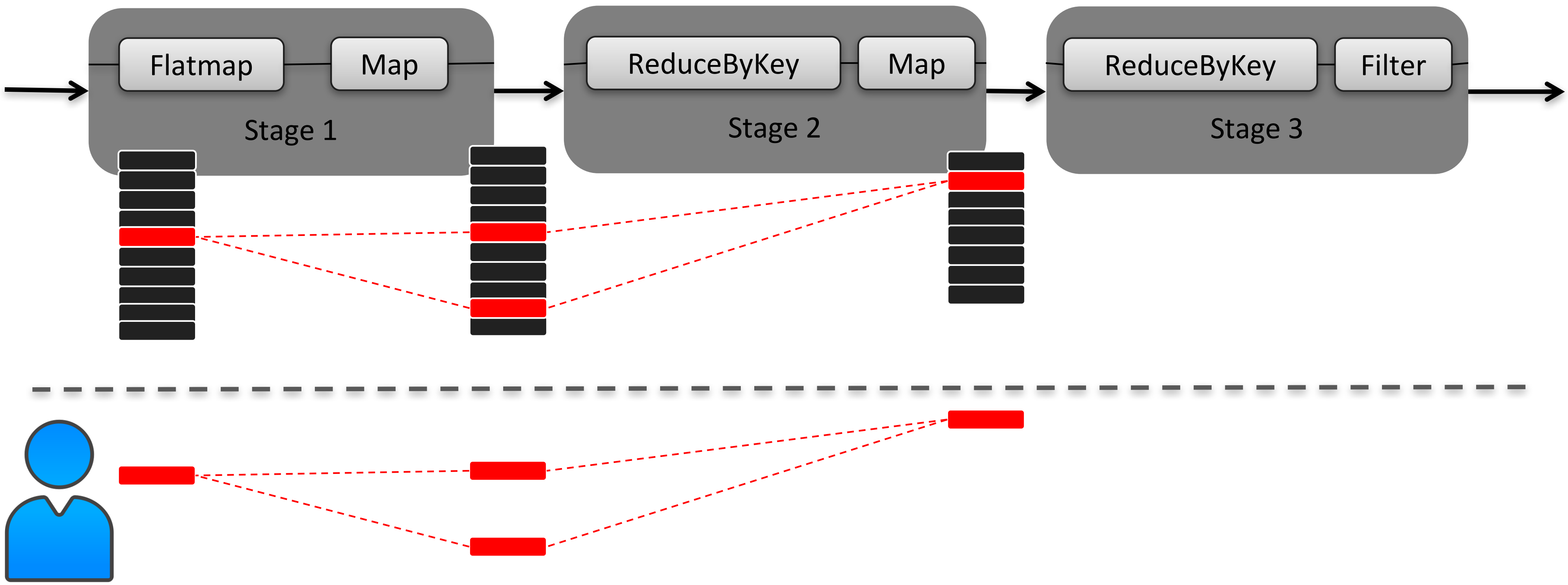
Watchpoint captures individual data records matching a user-provided guard

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



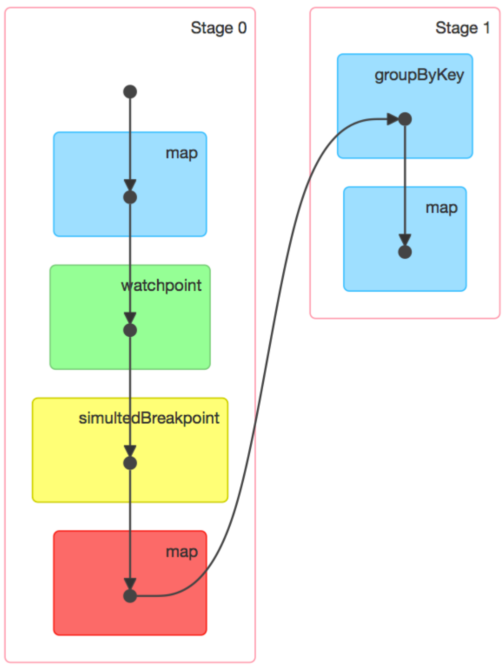
A user can also issue tracing queries on intermediate records at realtime

Demo: BigDebug Interactive Debugger
[FSE 2016 Demo, SIGMOD 2017 Demo]

Breakpoint Controls

Resume | Step Over

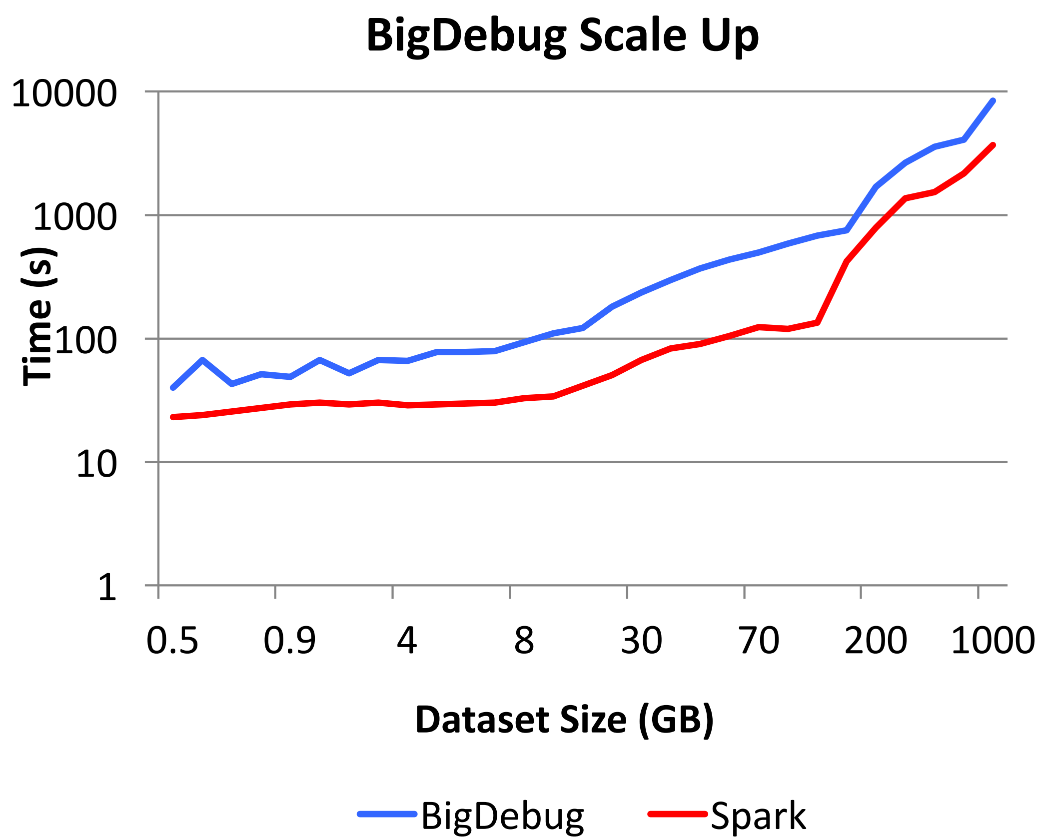
Current Breakpoint location is after the simultedBreakpoint at AliceStudentAnalysis.scala:24



AliceStudentAnalysis.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

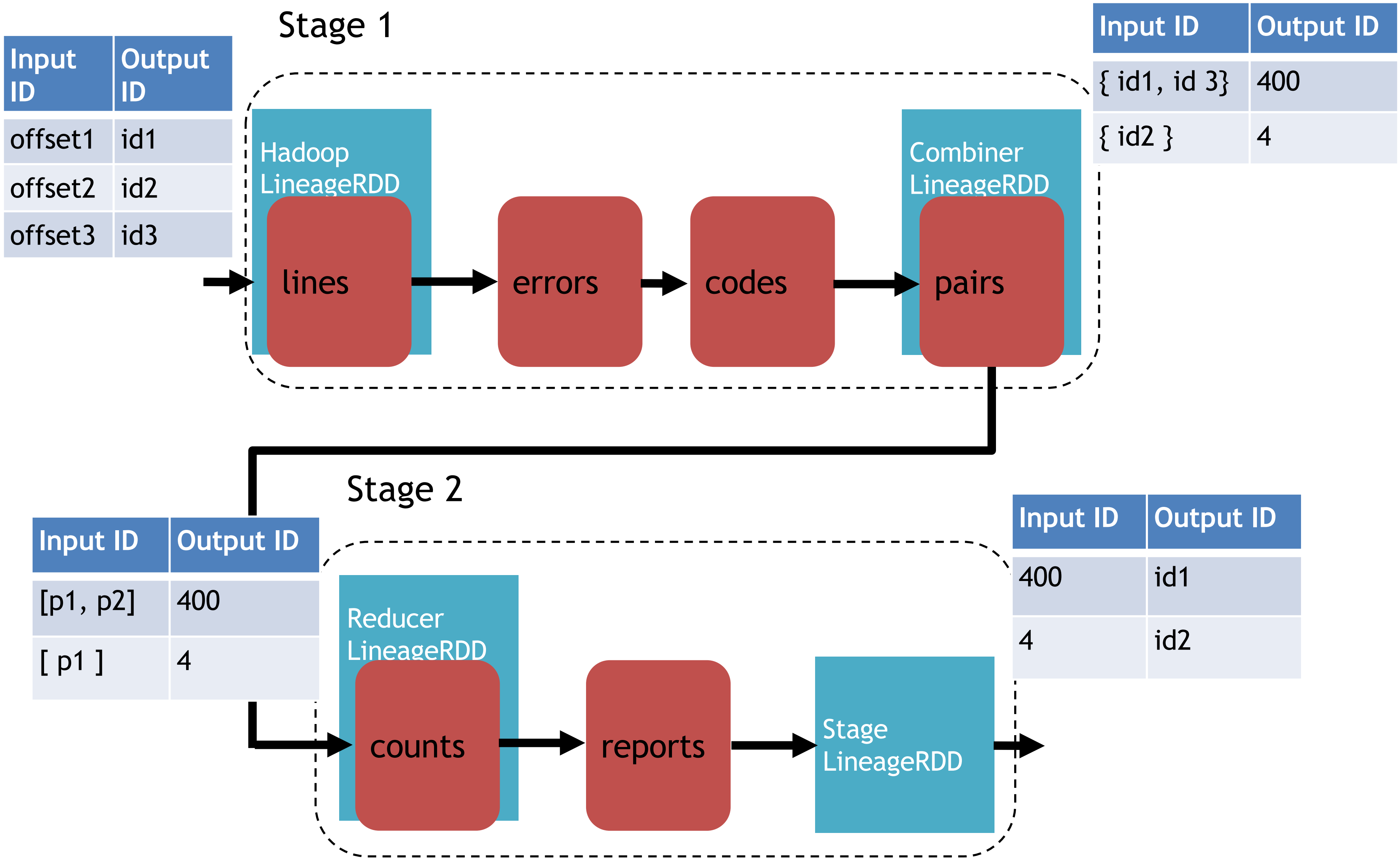
```
SELECT time, AVG(temp)
    FROM sensors
    GROUP BY time
```

Sensors			
Tuple-ID	Time	Sendor-ID	Temperature
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 ⋈ 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]

UCLA

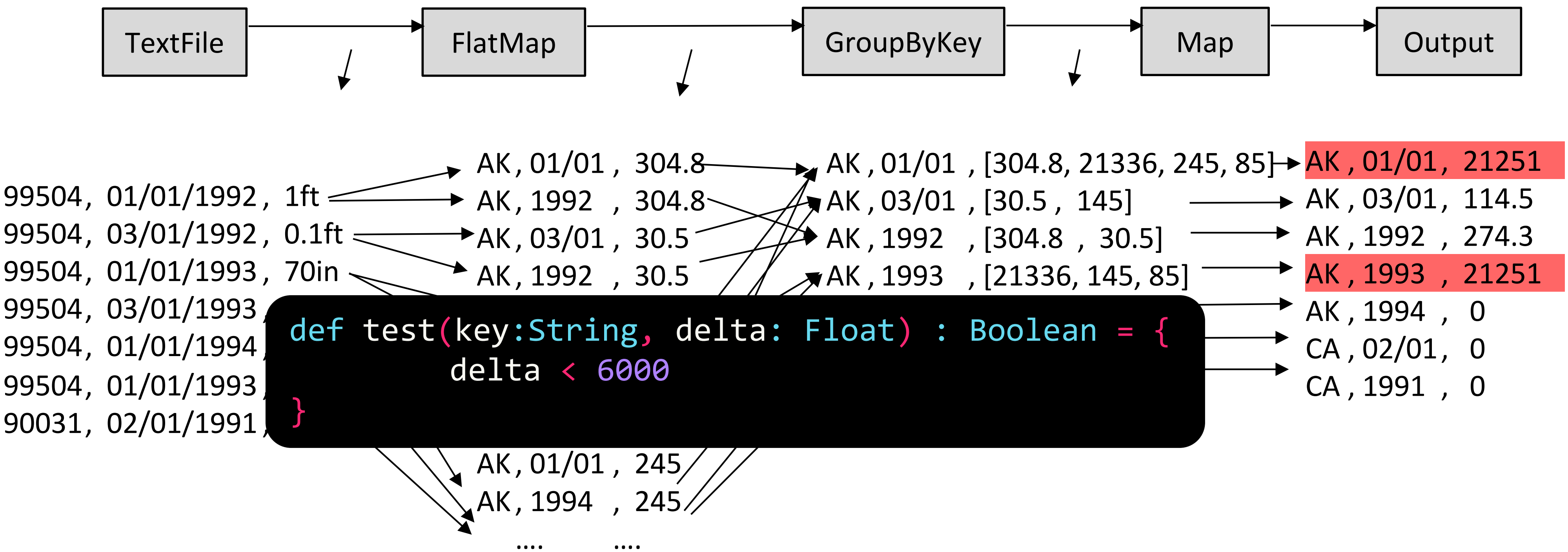
Motivating Example

- 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

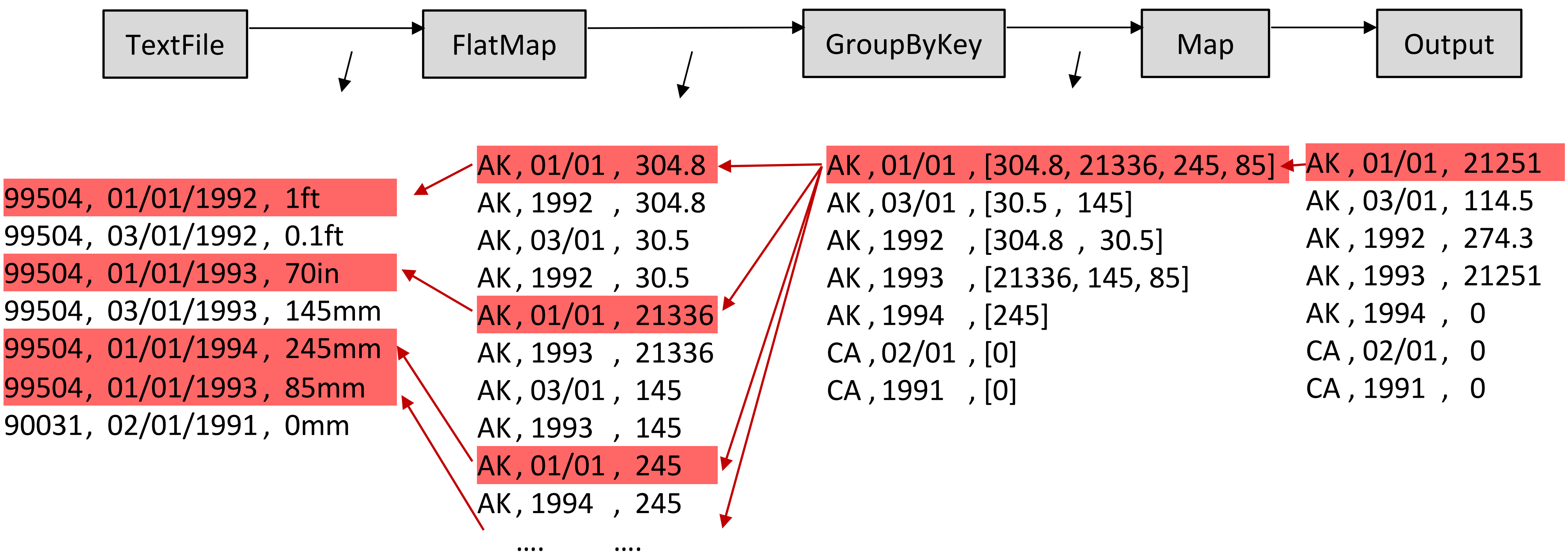
Problem Definition

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



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

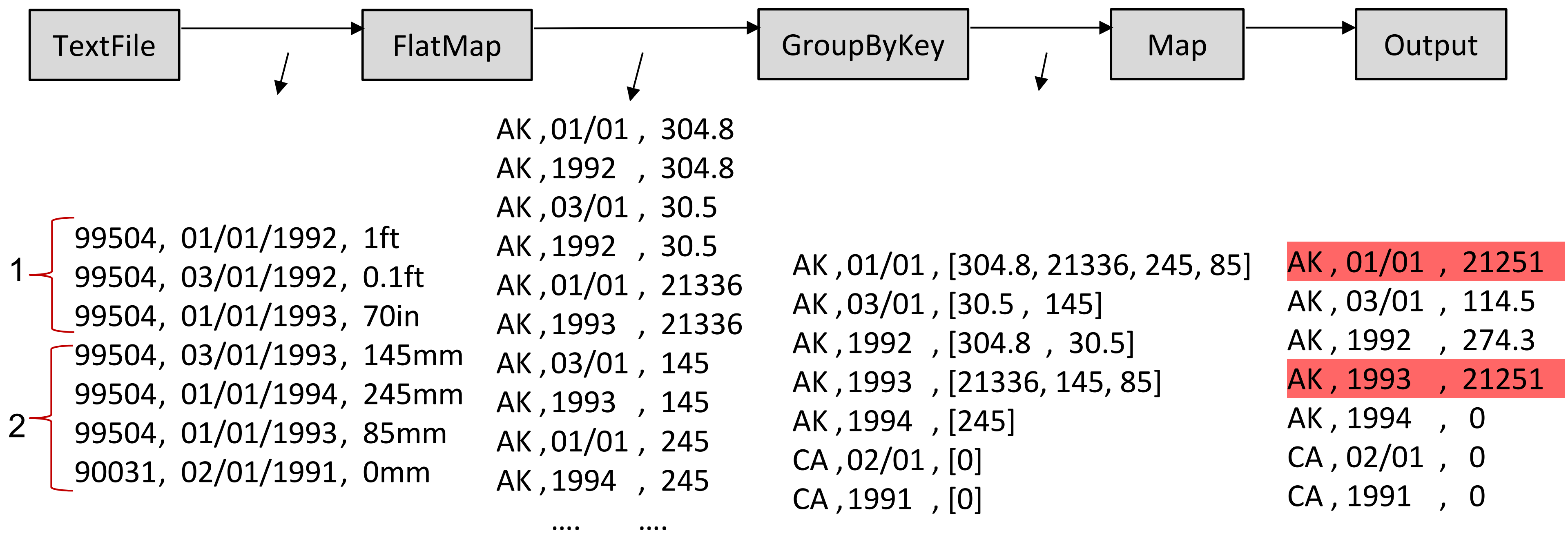
Existing Approach 1: Data Provenance for Spark



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

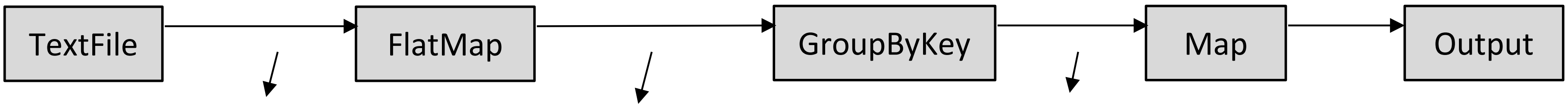
- Delta Debugging performs a systematic binary search-like procedure on the input dataset using a test oracle function



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



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]

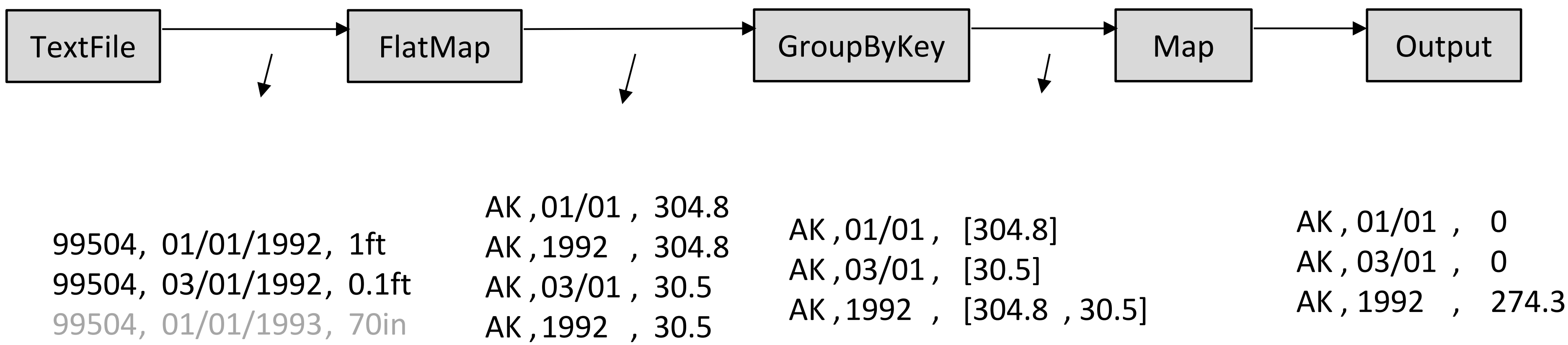
AK , 01/01 , 21031
AK , 03/01 , 0
AK , 1992 , 274.3
AK , 1993 , 0

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

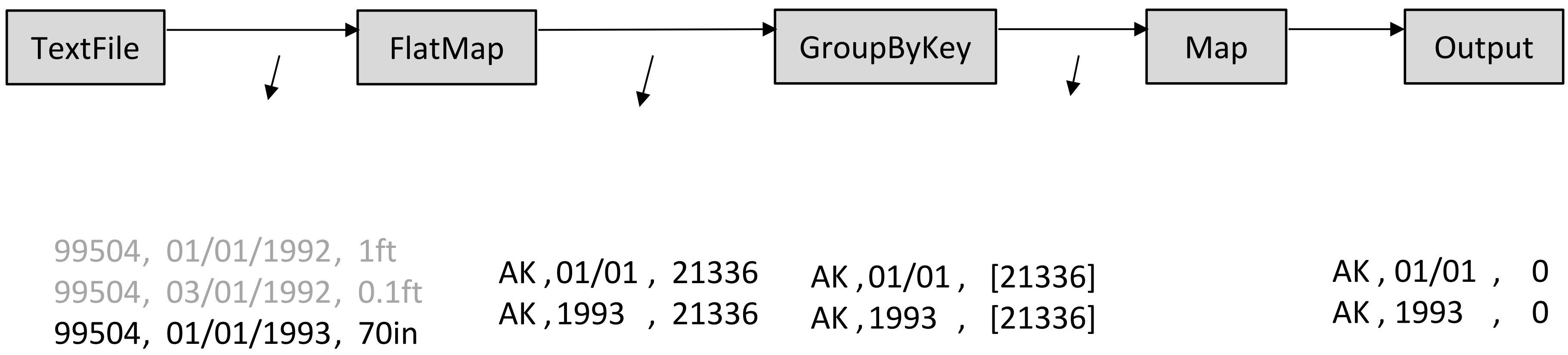


Run 3

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

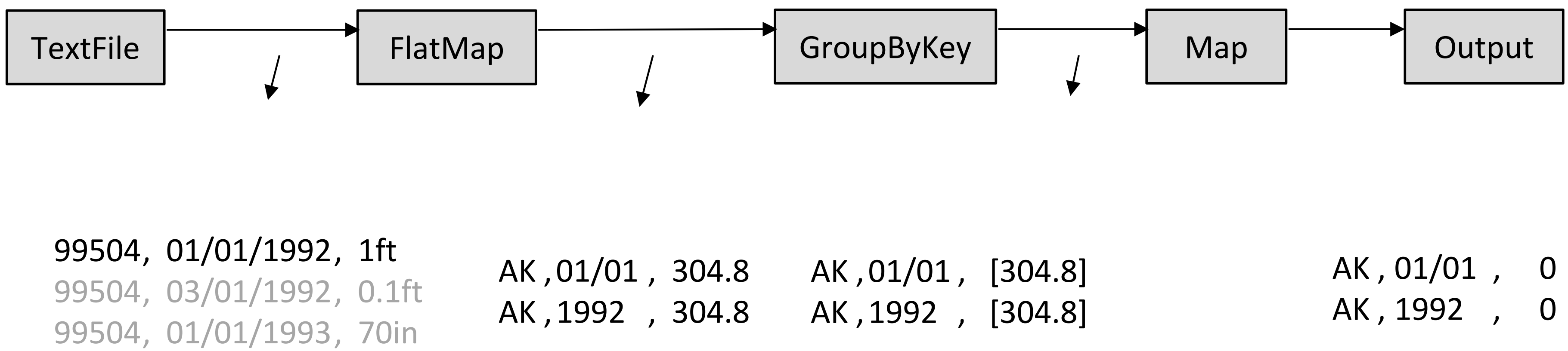


Run 4

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

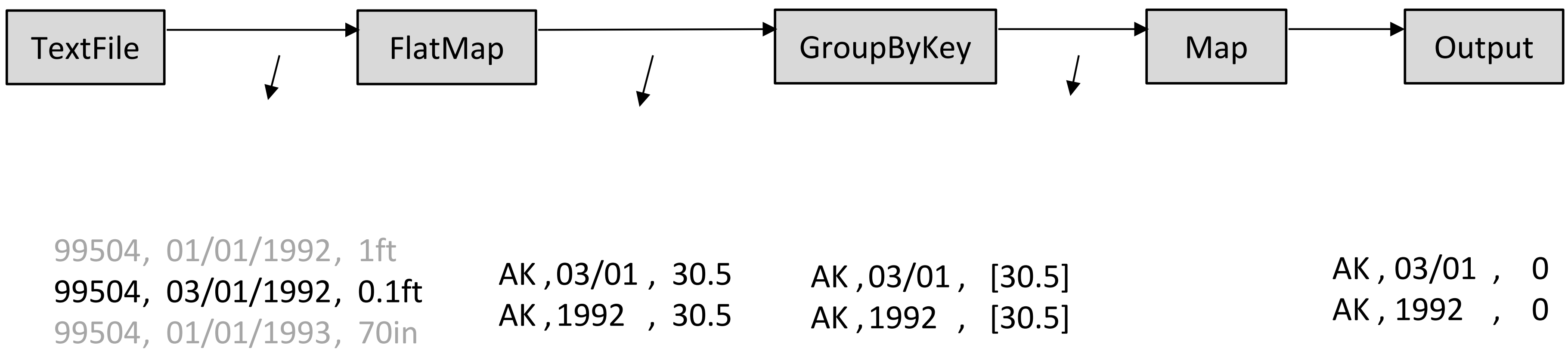


Run 5

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

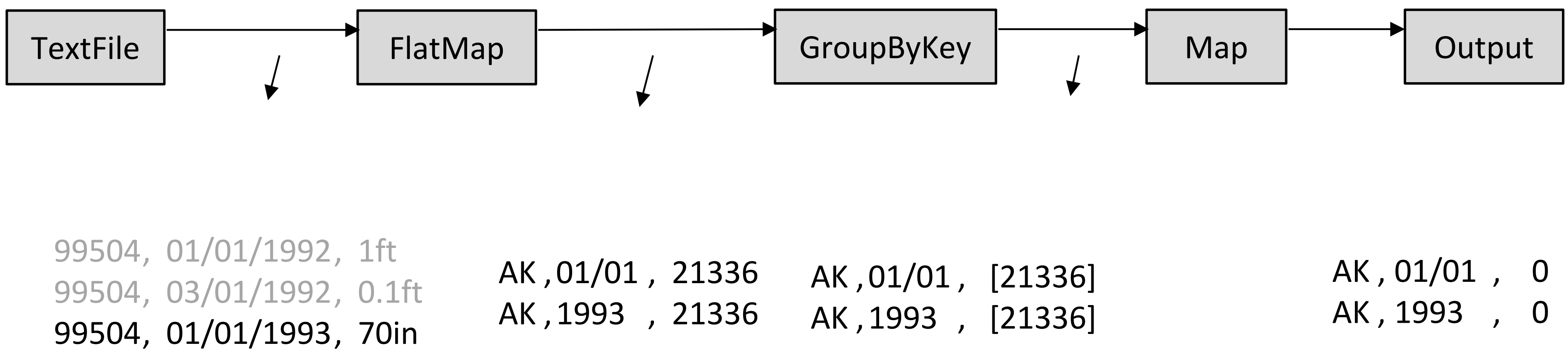


Run 6

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

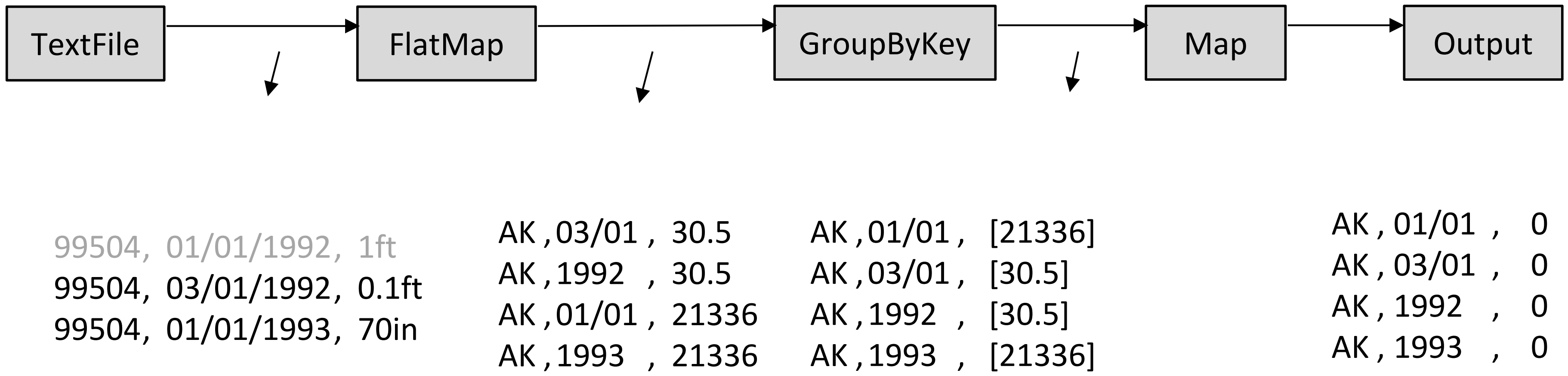


Run 7

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

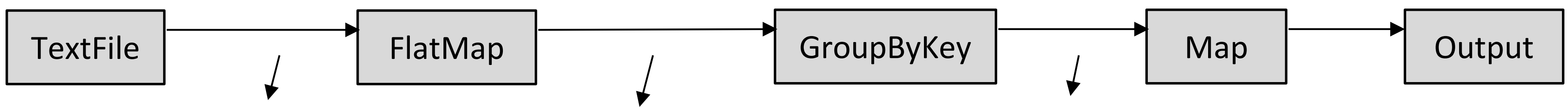


Run 8

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



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

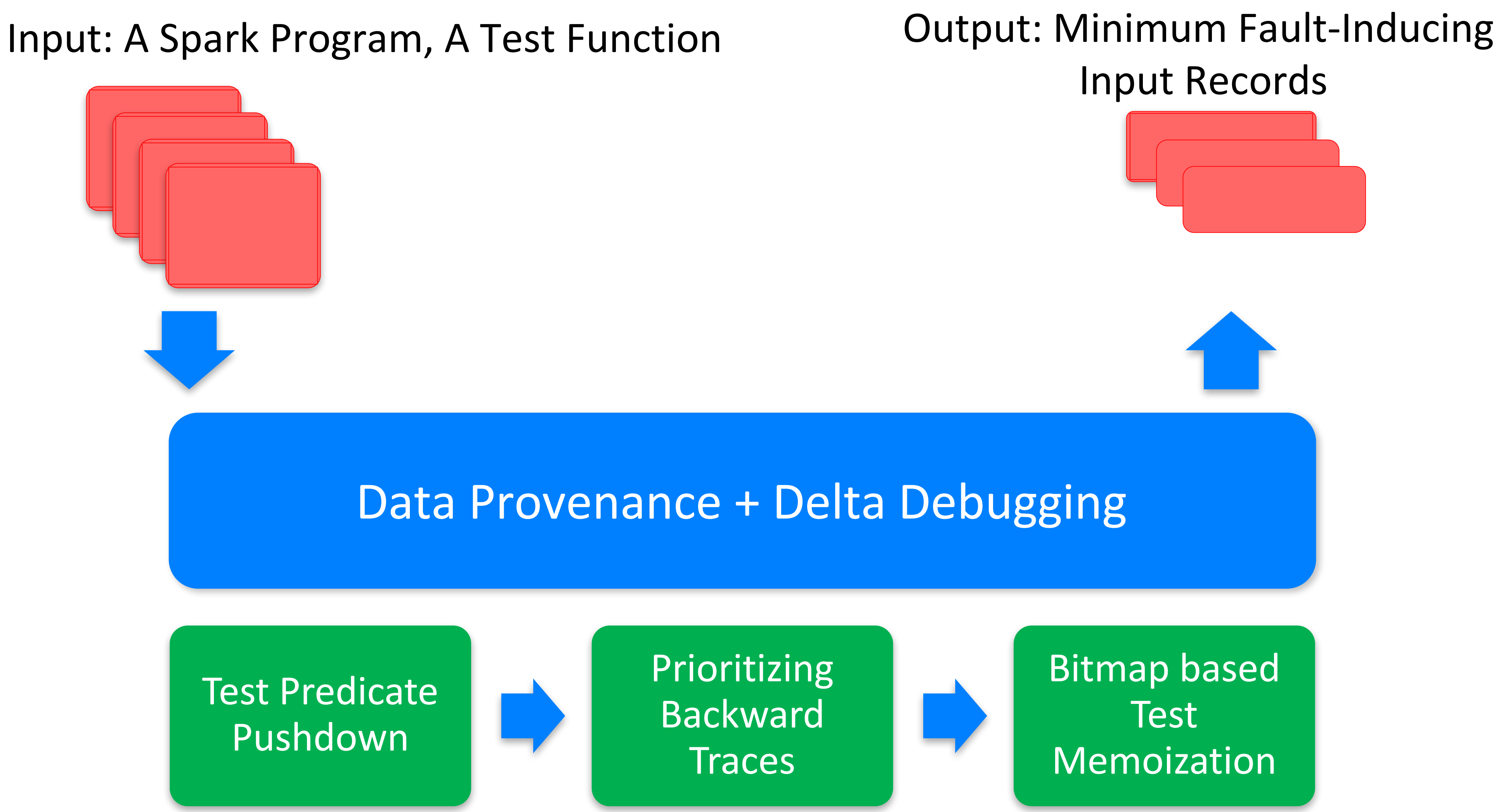
AK , 01/01 , [304.8, 21336]
AK , 1992 , [304.8]
AK , 1993 , [21336]

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

Run 9

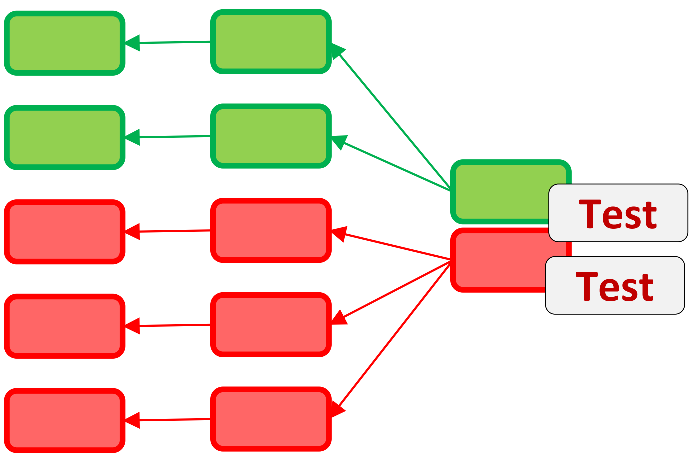
It does not prune input records known to be irrelevant because of the lack of semantic understanding of data-flow operators

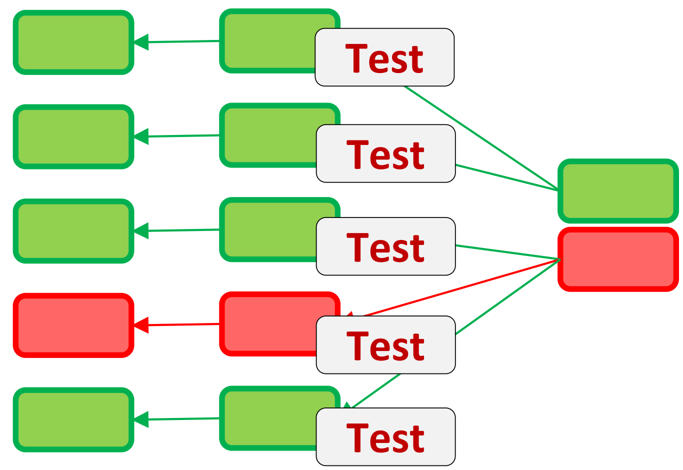
Automated Debugging in DISC with BigSift



Optimization 1: Test Predicate Pushdown

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

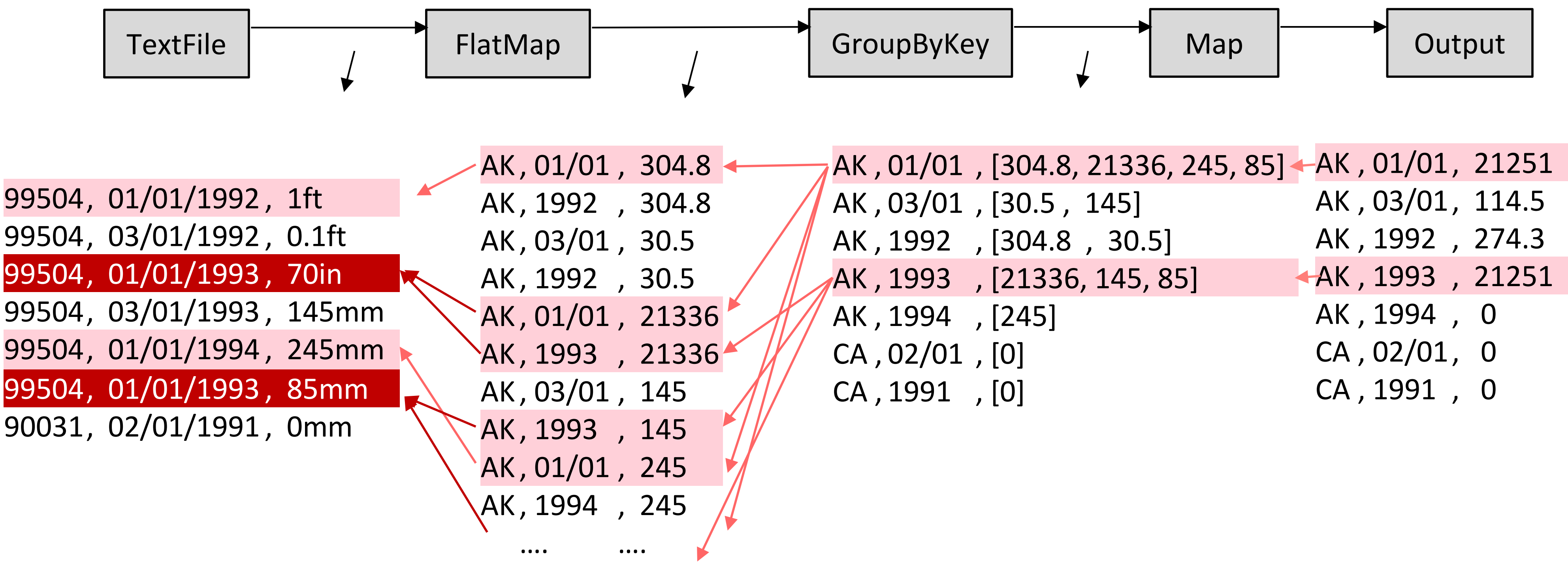




If applicable, BigSift pushes down the test function to test the output of combiners in order to isolate the faulty partitions.

Optimization 2: Prioritizing Backward Traces

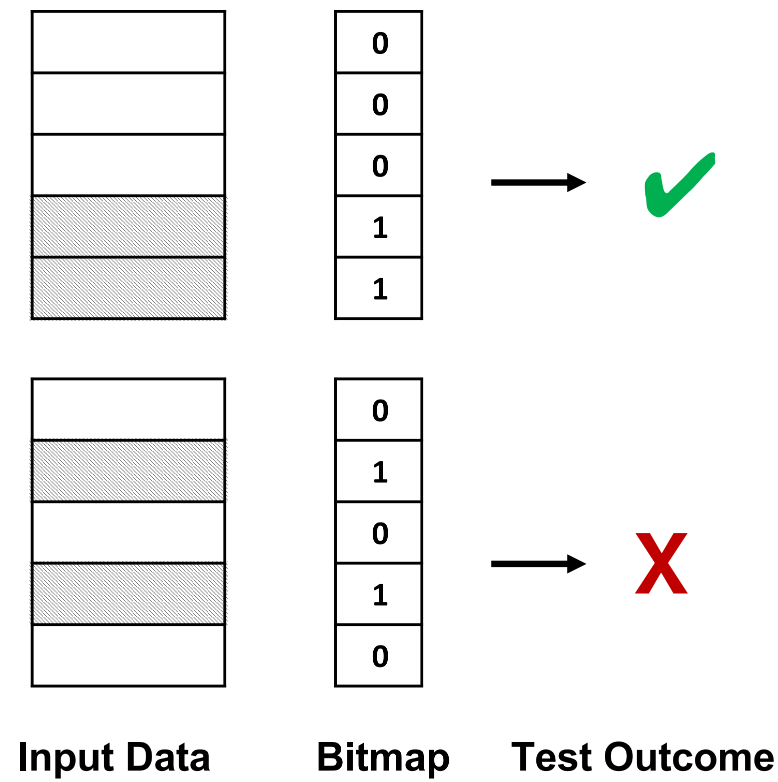
- **Observation:** The same faulty input record may contribute to multiple output records failing the test.



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

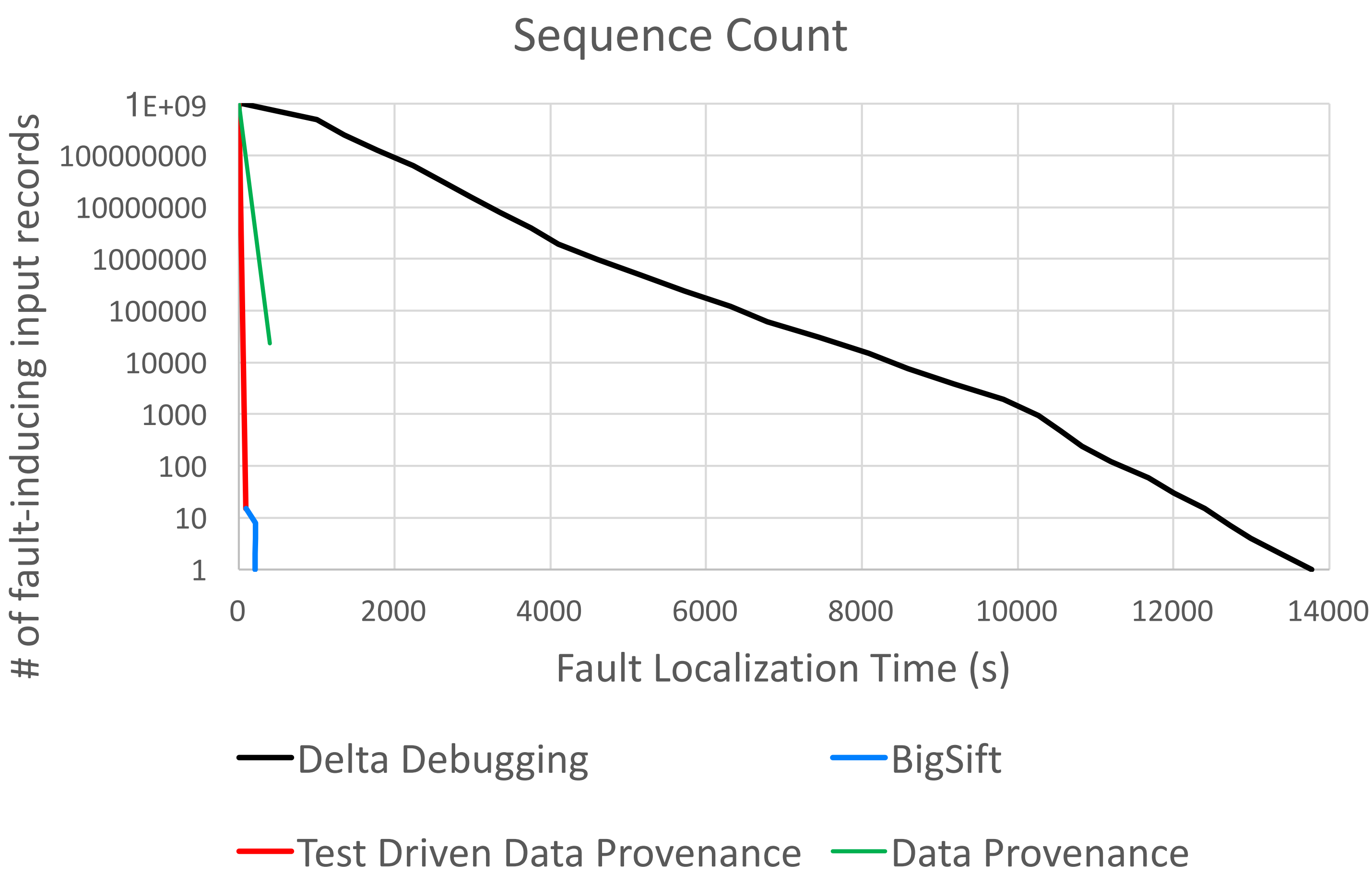
BigSift provides up to a 66X speed up in isolating the precise fault-inducing input records, in comparison to the baseline DD

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

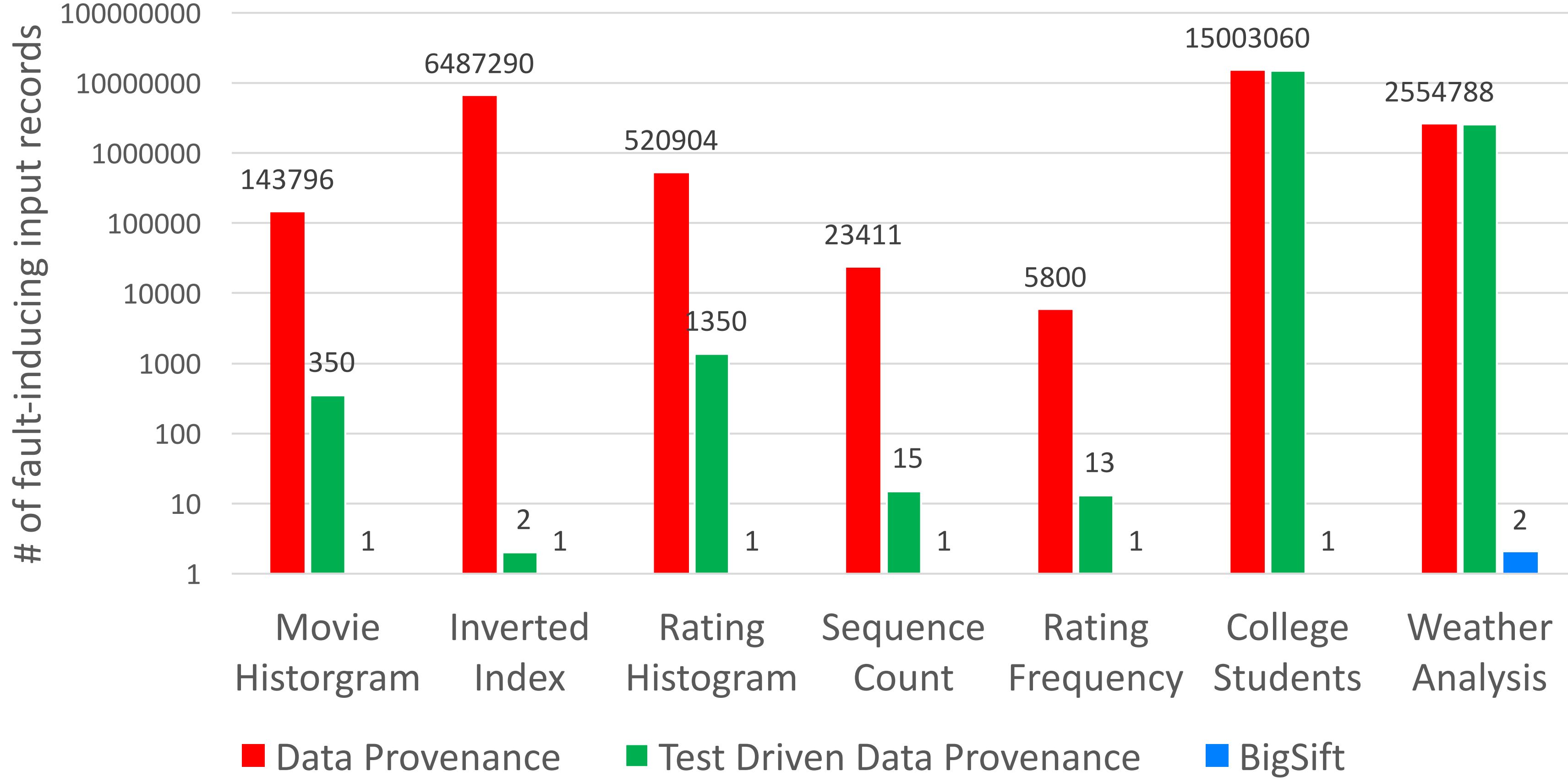
On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

RQ2: Debugging Time



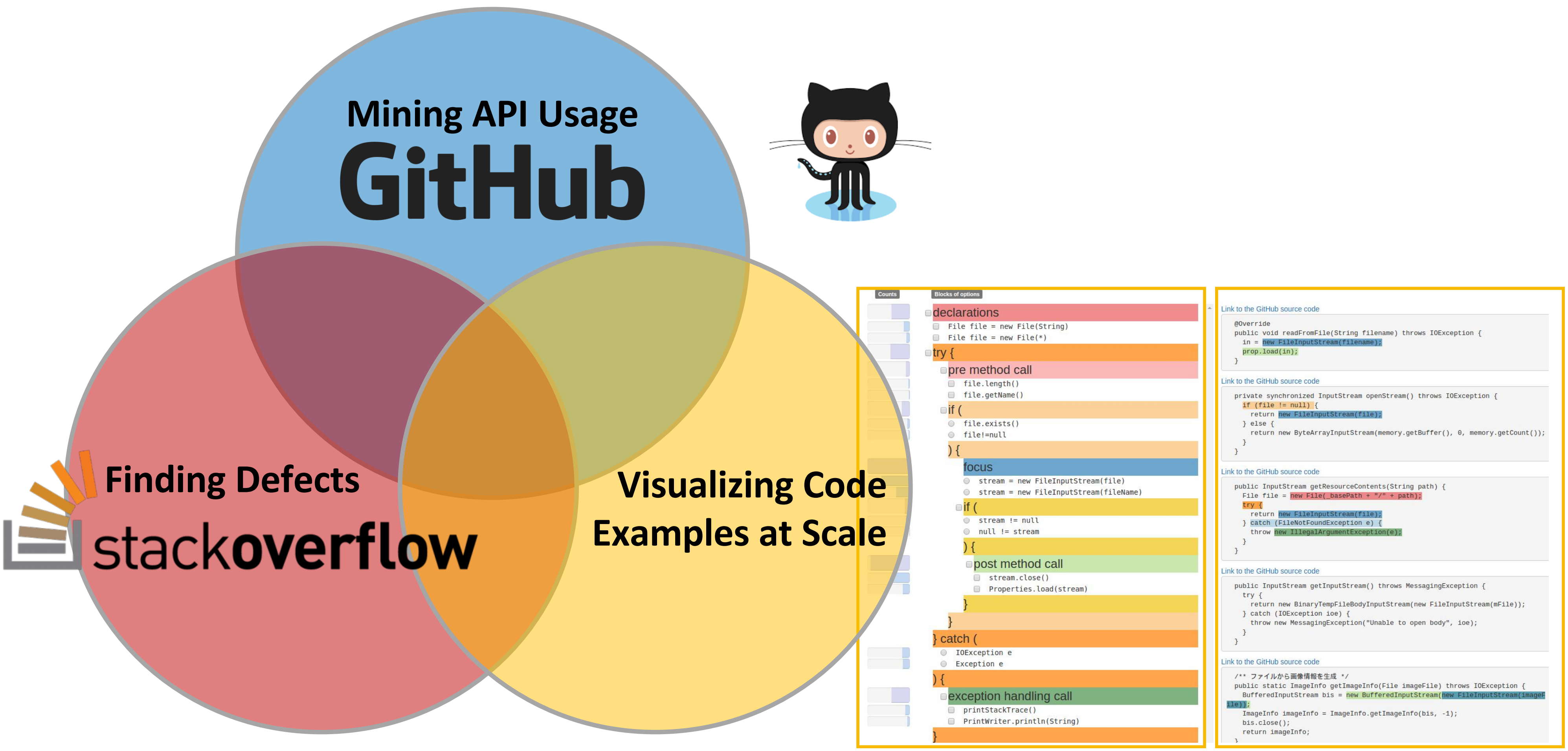
On average, BigSift takes 62% less time to debug a single faulty output than the time taken for a single run on the entire data.

RQ3: Fault Localizability over Data Provenance



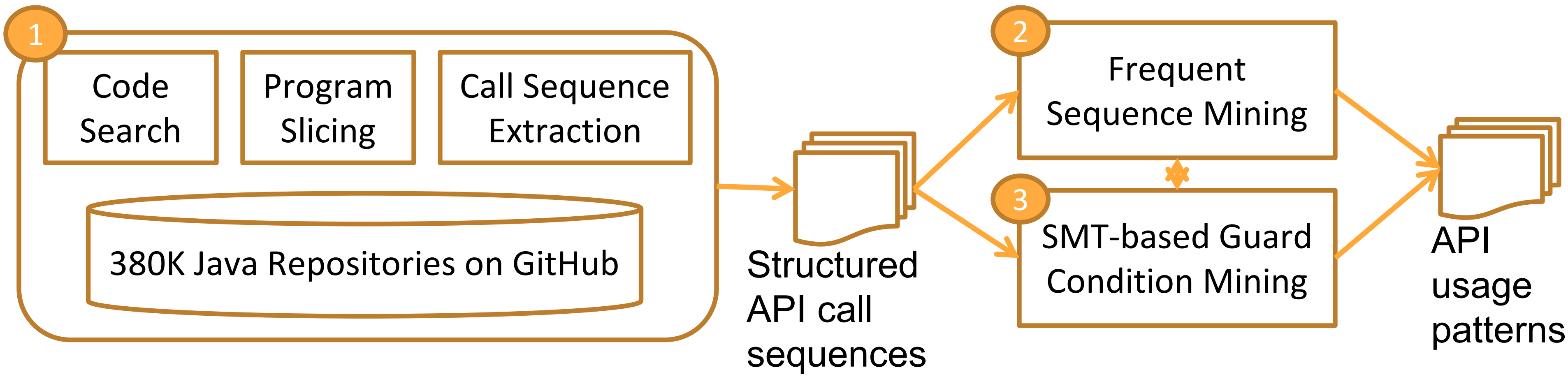
BigSift leverages DD after DP to continue fault isolation, achieving several orders of magnitude 10^3 to 10^7 better precision

Part 2:
Data Science *elevating* Software Engineering



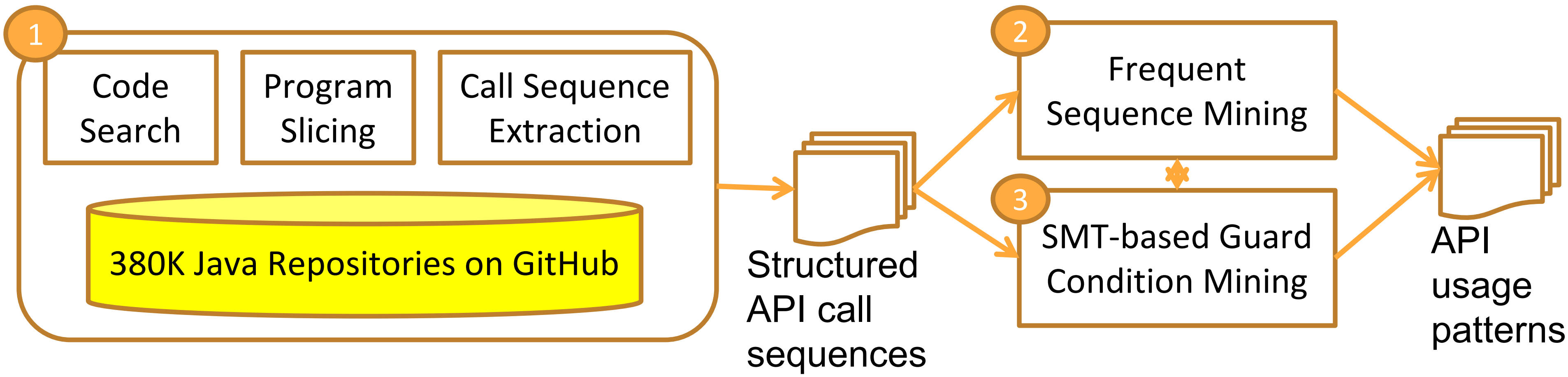
API Usage Mining from GitHub [ICSE 2018]

We contrast SO snippets with API usage patterns mined from 380K GitHub projects.



Insight 1: Mining a Large Code Corpus

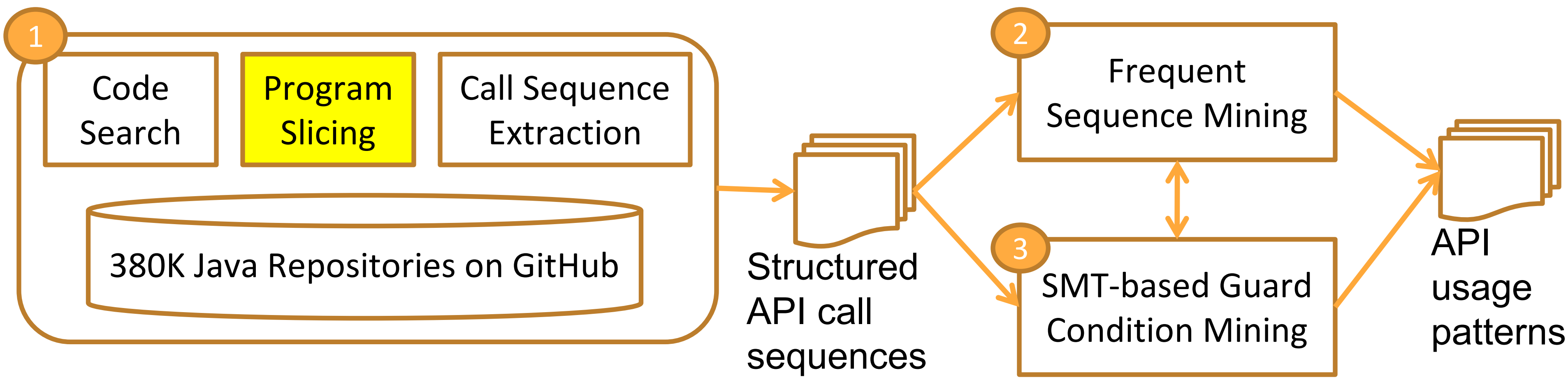
Our code corpus includes 380K GitHub projects with at least 100 revisions and 2 contributors.



Dyer et al. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. ICSE 2013.

Insight 2: Removing Irrelevant Statements via Program Slicing

We perform backward and forward slicing to identify data- and control-dependent statements to an API method of interest.



```java
void initInterfaceProperties(String temp, File dDir) {
  if(!temp.equals("props.txt")) {
     log.error("Wrong Template.");
     return;
  }
  // load default properties
  FileInputStream in = new FileInputStream(temp);
  Properties prop = new Properties();
  prop.load(in);
  ... init properties ...
  // write to the property file
  String fPath=dDir.getAbsolutePath()+"/interface.prop";
  File file = new File(fPath);
  if(!file.exists()) {
    file.createNewFile();
  }
  FileOutputStream out = new FileOutputStream(file);
  prop.store(out, null);
  in.close();
}
```

GitHub example of `File.createNewFile`

The focal API method →

```
void initInterfaceProperties(String temp, File dDir) {
  if(!temp.equals("props.txt")) {
    log.error("Wrong Template.");
    return;
  }
  // load default properties
  FileInputStream in = new FileInputStream(temp);
  Properties prop = new Properties();
  prop.load(in);
  ... init properties ...
  // write to the property file
  String fPath=dDir.getAbsolutePath()+"/interface.prop";
  File file = new File(fPath);
  if(!file.exists()) {
    file.createNewFile();
  }
  FileOutputStream out = new FileOutputStream(file);
  prop.store(out, null);
  in.close();
}
```

Data dependency up to **one** hop, i.e., direct dependency

The focal API method

control

data

```
void initInterfaceProperties(String temp, File dDir) {
  if(!temp.equals("props.txt")) {
     log.error("Wrong Template.");
     return;
  }
  // load default properties
  FileInputStream in = new FileInputStream(temp);
  Properties prop = new Properties();
  prop.load(in);
  ... init properties ...
  // write to the property file
  String fPath=dDir.getAbsolutePath()+"/interface.prop";
  File file = new File(fPath);
  if(!file.exists()) {
    file.createNewFile();
  }
  FileOutputStream out = new FileOutputStream(file);
  prop.store(out, null);
  in.close();
}
```

Data dependency up to two hops

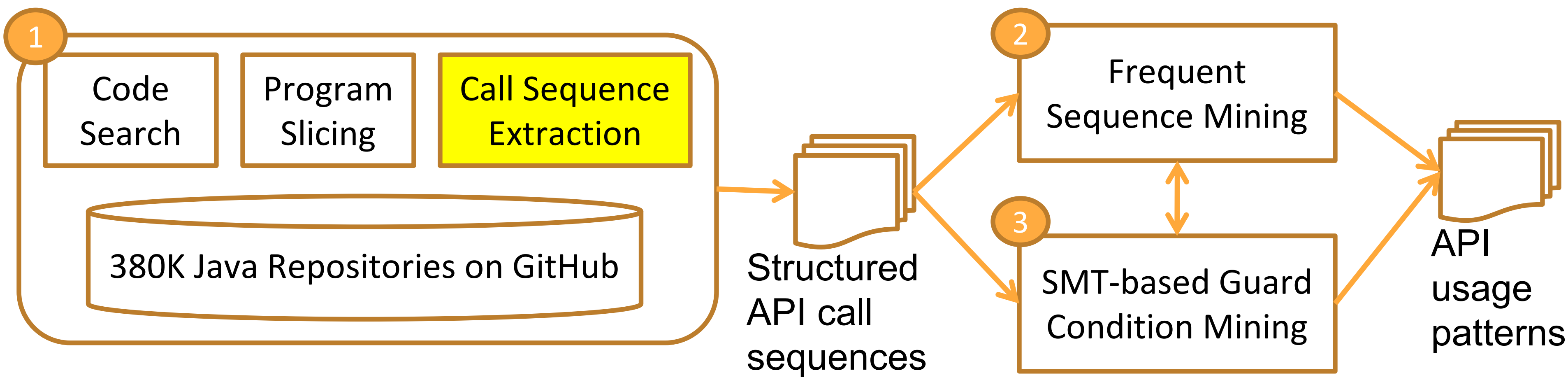
The focal API method

control

data

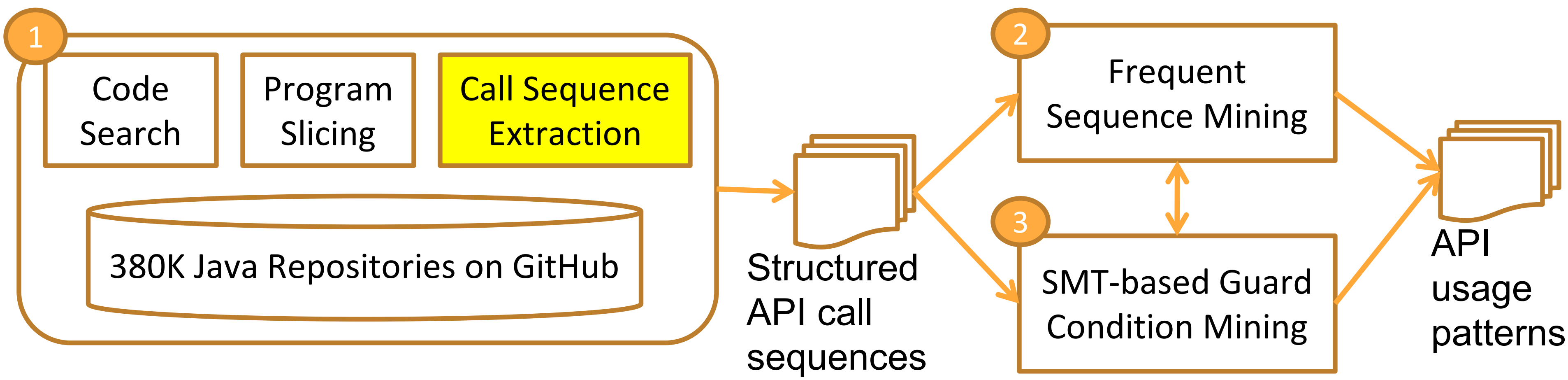
Insight 3: Capture Semantics Info in API Usage

It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.



Insight 3: Capture Semantics Info in API Usage

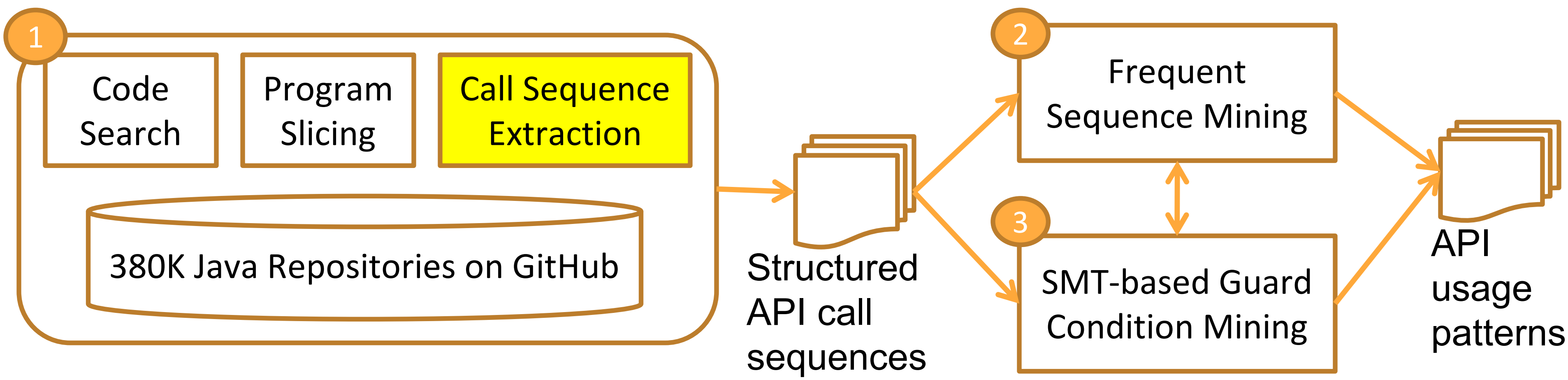
It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.




```
new File (String); try {; new FileInputStream(File)@arg0.exists(); } catch (IOException) {; }
```

Insight 3: Capture Semantics Info in API Usage

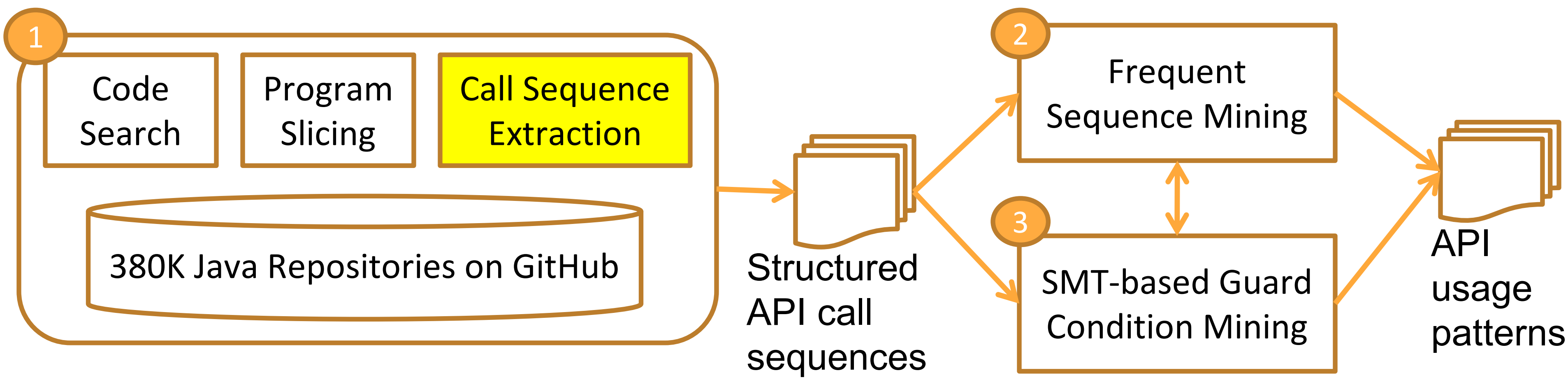
It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.




```
new File (String); try {; new FileInputStream(File)@arg0.exists(); } catch (IOException) {; }
```

Insight 3: Capture Semantics Info in API Usage

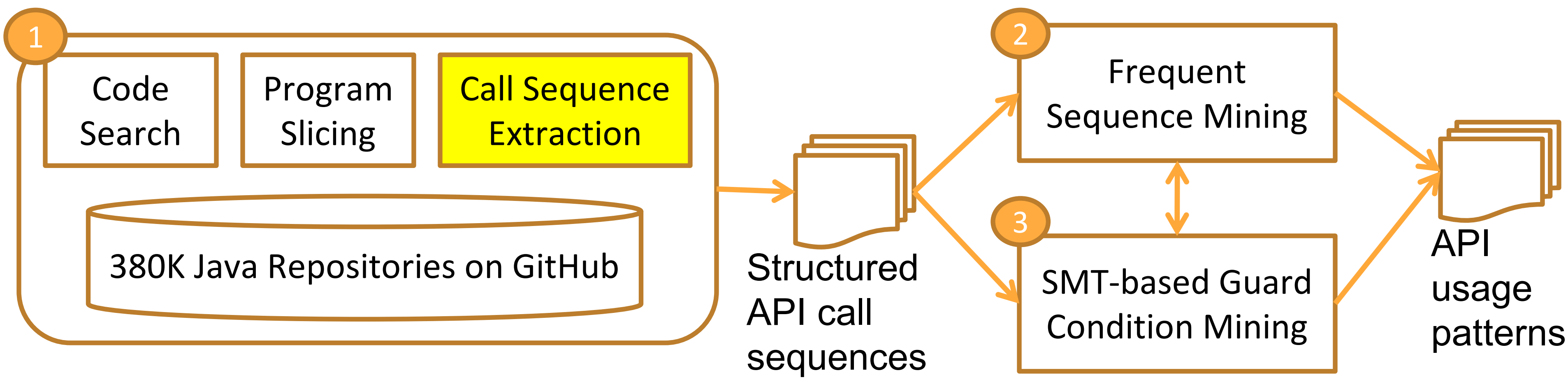
It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.




```
new File (String); try {; new FileInputStream(File)@arg0.exists(); } catch (IOException) {; }
```

Insight 3: Capture Semantics Info in API Usage

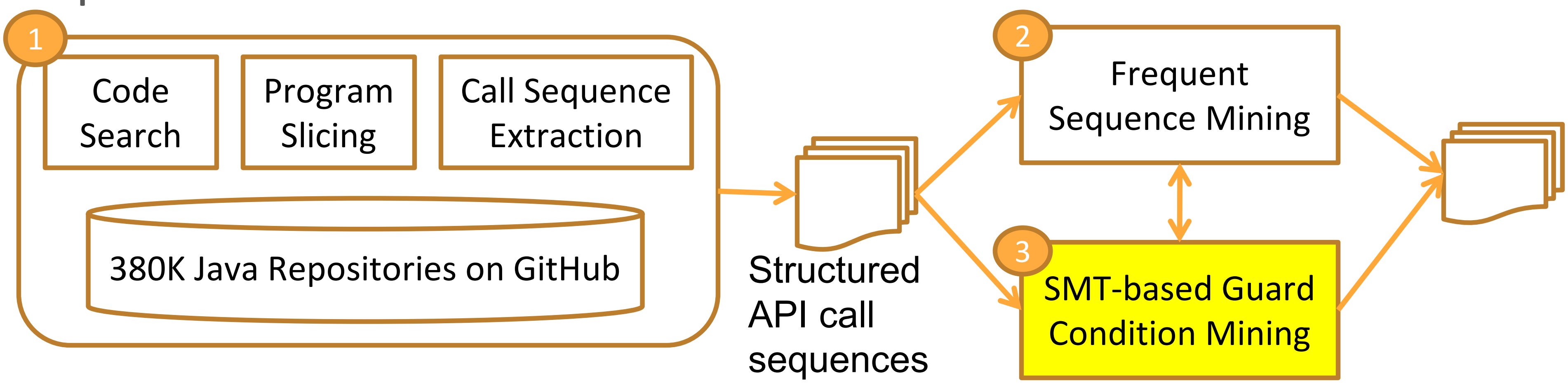
It is important to capture the temporal ordering, enclosing control structures, and appropriate guard conditions of API calls.




```
new File (String); try {; new FileInputStream(File)@arg0.exists(); } catch (IOException) {; }
```

Insight 4: Variations in Guard Conditions

Guard conditions are canonicalized and grouped based on logical equivalence.

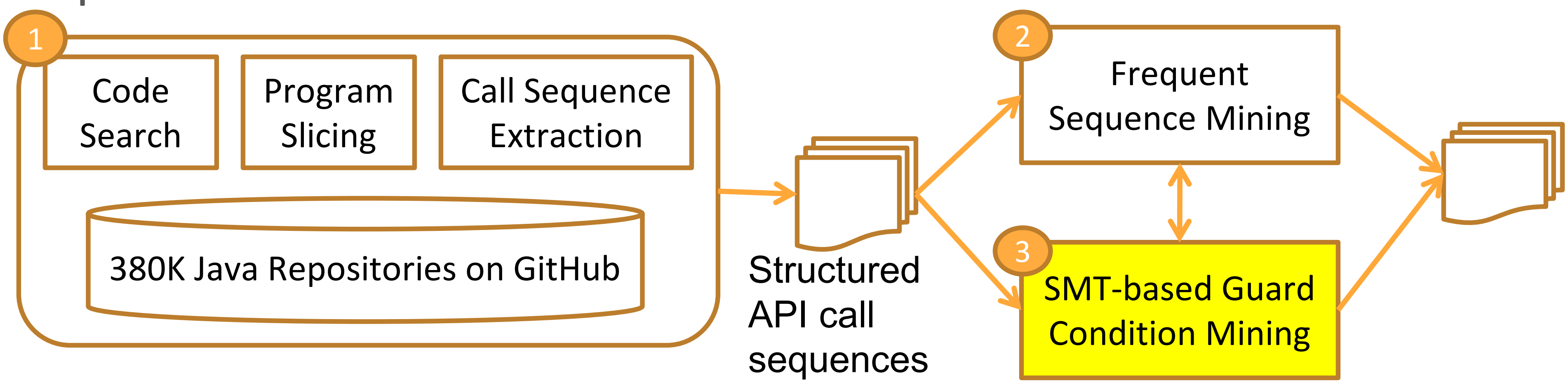


Two equivalent guard conditions for `String.substring`:

`arg0>=0 && arg0<=rcv.length()` ⇔ `arg0>-1 && arg0<rcv.length()+1`

Insight 4: Variations in Guard Conditions

Guard conditions are canonicalized and grouped based on logical equivalence.

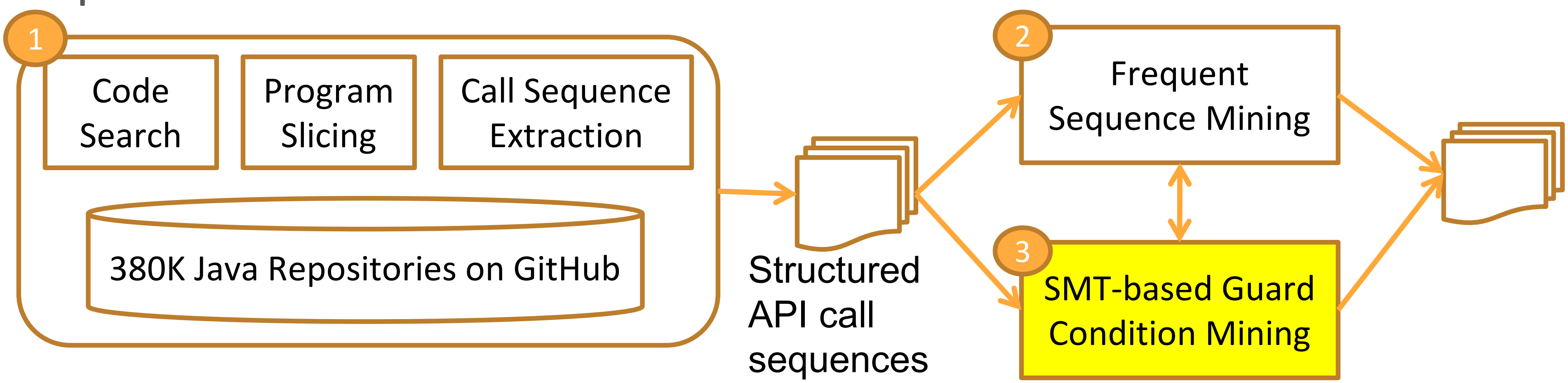


Two equivalent guard conditions for `String.substring`:

`arg0>=0 && arg0<=rcv.length()` ⇔ `arg0>-1 && arg0<rcv.length()+1`

Insight 4: Variations in Guard Conditions

Guard conditions are canonicalized and grouped based on logical equivalence.



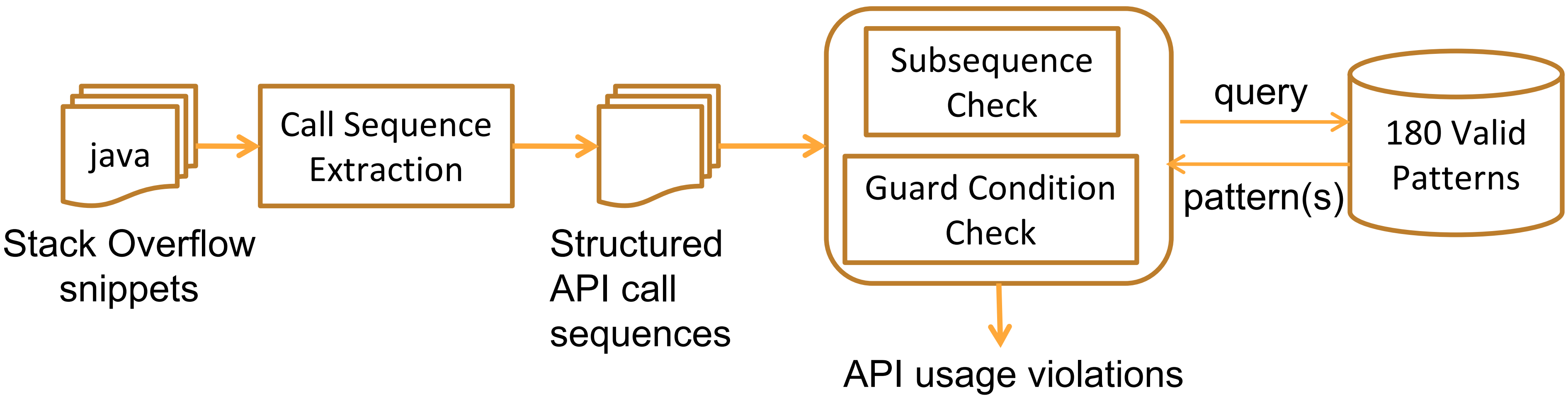
Two equivalent guard conditions for `String.substring`:

`arg0>=0 && arg0<=rcv.length()` ⇔ `arg0>-1 && arg0<rcv.length()+1`

API Misuse Detection in StackOverflow

We examine 220K SO posts with 180 confirmed patterns.

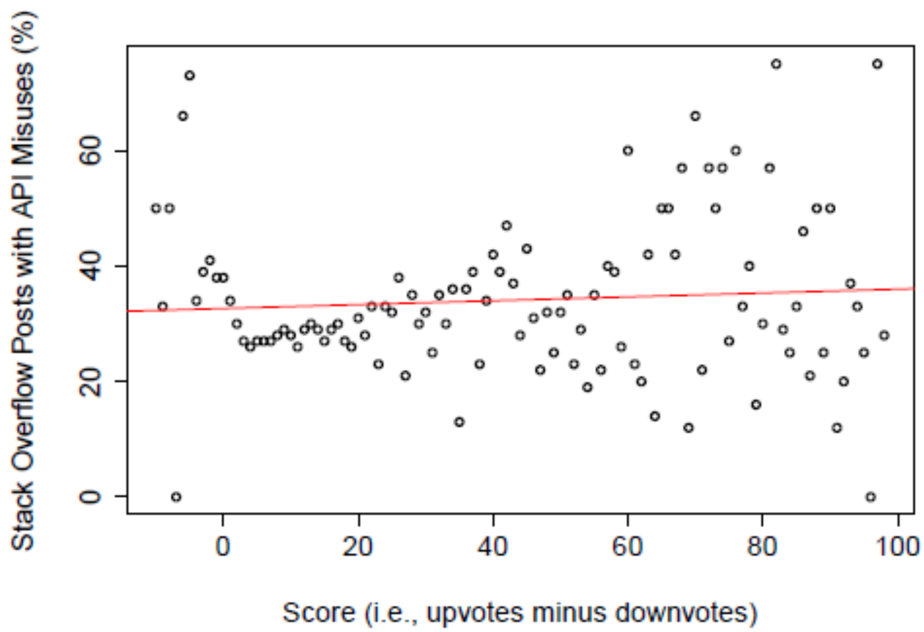
=> 31% of SO posts contain API usage violations!



Dataset: http://web.cs.ucla.edu/~tianyi.zhang/examplecheck.html

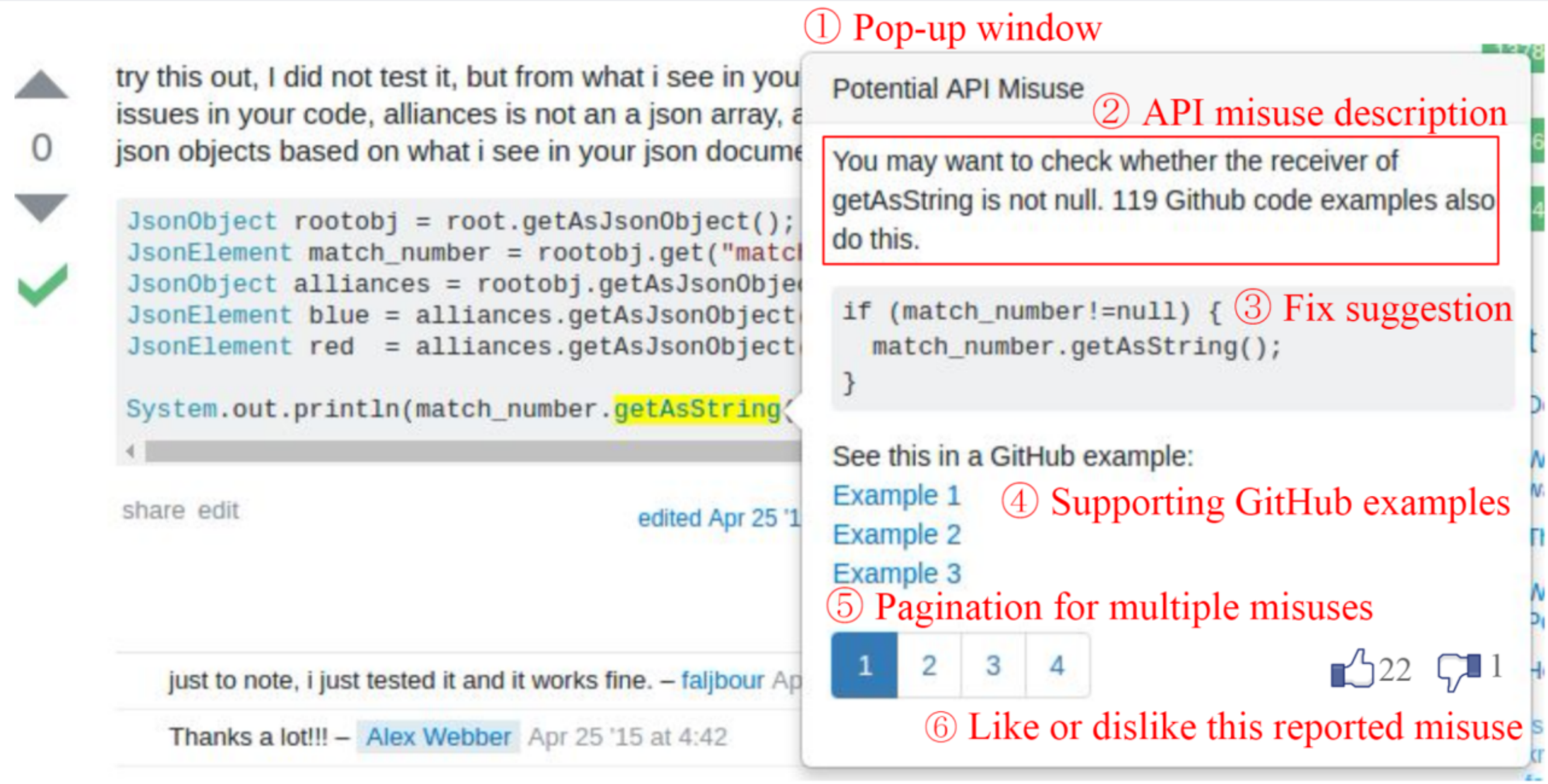
API Misuses are Prevalent on Stack Overflow

Highly-voted posts are not necessarily more reliable in terms of correct API usage.



Network, database, IO, crypto, string manipulation APIs are more likely to be misused.

ExampleCheck: Augmenting Stack Overflow with API Usage Patterns Mined from GitHub



Add Chrome Logo

chrome web store

miryungkimho@gmail.com

Home › Extensions › ExampleCheck

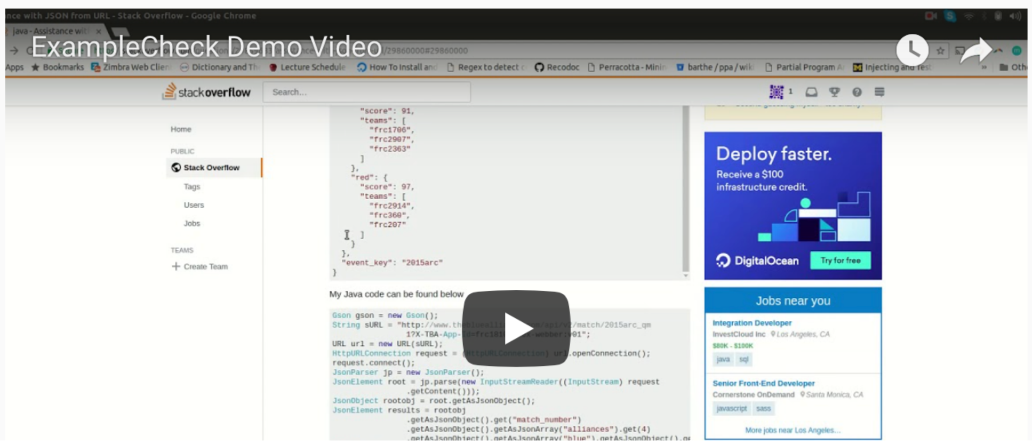
ExampleCheck

Offered by: Tianyi Zhang

0 | Developer Tools | 19 users

Available on Chrome

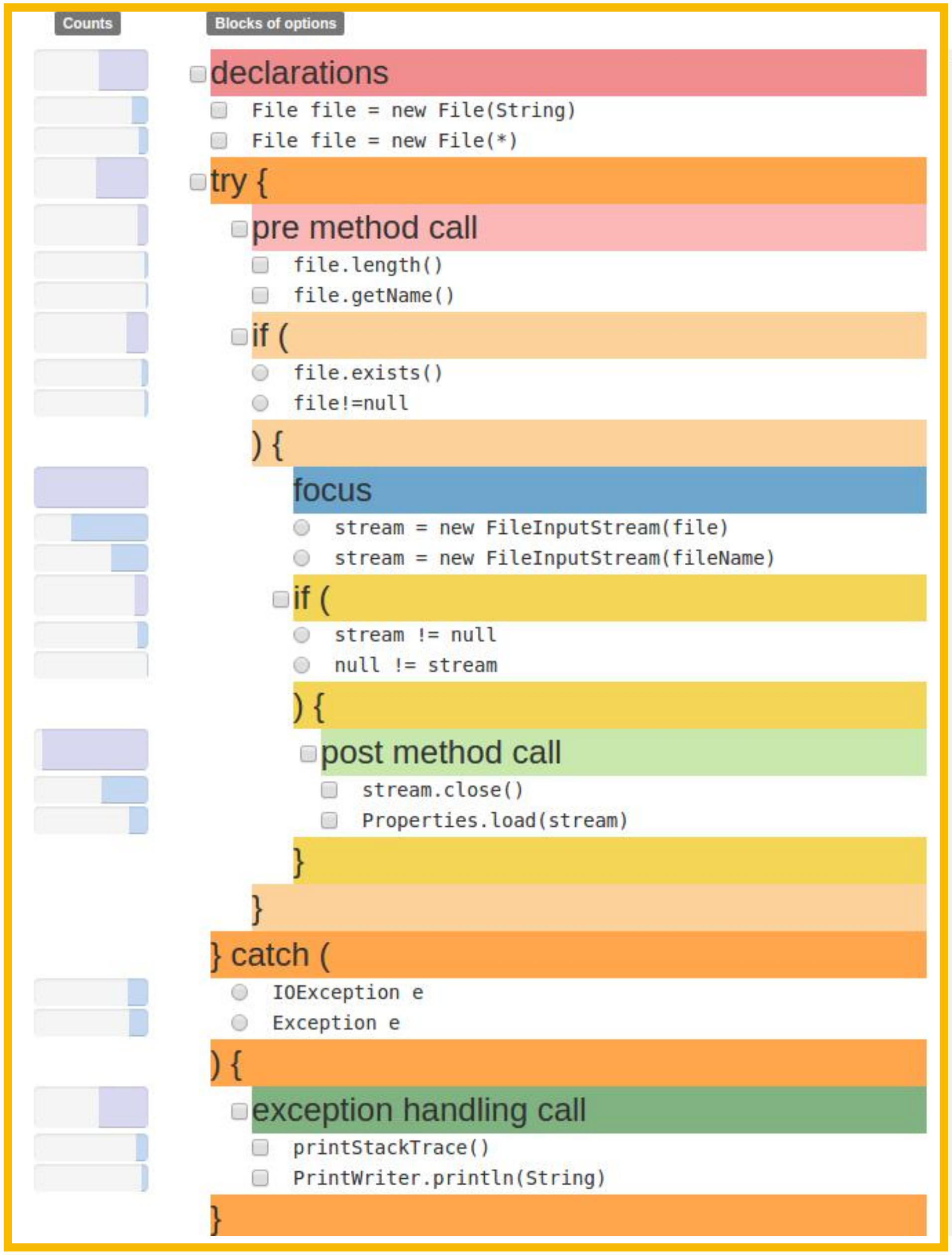
Overview Reviews Related



https://chrome.google.com/webstore/detail/examplecheck/amliempebckaiaklimcpopomlnklkioe

Examplore: Visualizing Code Examples at Scale [CHI 2018]

Examplore visualizes hundreds of code examples using the same API.


Counts
Blocks of options
declarations
File file = new File(String)
File file = new File(*)
try {
pre method call
file.length()
file.getName()
if (
file.exists()
file!=null
) {
focus
stream = new FileInputStream(file)
stream = new FileInputStream(fileName)
if (
stream != null
null != stream
) {
post method call
stream.close()
Properties.load(stream)
}
}
} catch (
IOException e
Exception e
) {
exception handling call
printStackTrace()
PrintWriter.println(String)
}


Link to the GitHub source code
@Override
public void readFromFile(String filename) throws IOException {
in = new FileInputStream(filename);
prop.load(in);
}
Link to the GitHub source code
private synchronized InputStream openStream() throws IOException {
if (file != null) {
return new FileInputStream(file);
} else {
return new ByteArrayInputStream(memory.getBuffer(), 0, memory.getCount());
}
}
Link to the GitHub source code
public InputStream getResourceContents(String path) {
File file = new File(_basePath + "/" + path);
try {
return new FileInputStream(file);
} catch (FileNotFoundException e) {
throw new IllegalArgumentException(e);
}
}
Link to the GitHub source code
public InputStream getInputStream() throws MessagingException {
try {
return new BinaryTempFileBodyInputStream(new FileInputStream(mFile));
} catch (IOException ioe) {
throw new MessagingException("Unable to open body", ioe);
}
}
Link to the GitHub source code
/** ファイルから画像情報を生成 */
public static ImageInfo getImageInfo(File imageFile) throws IOException {
BufferedInputStream bis = new BufferedInputStream(new FileInputStream(imageFile));
ImageInfo imageInfo = ImageInfo.getImageInfo(bis, -1);
bis.close();
return imageInfo;
}

Thanks to my collaborators

UCLA on Big Data Debugging: Muhammad Ali Gulzar*, Tyson Condie, Matteo Interlandi, Mingda Li, Michael Han, Sai Deep Tetali, Todd Millstein

Microsoft Research on Data Scientist Studies: Tom Zimmermann, Andrew Begel, and Rob DeLine

UCLA, UC Berkeley, Iowa State on API Usage Mining:
Tianyi Zhang*, Elena Glassman, Bjorn Hartmann, Ganesha Upadhyaya, Anastasia Reinhart, Hridesh Rajan

Big Data needs **awesome software engineering tools**

Diagnose

- ✓ Debugging
- ✓ Intelligent sampling and testing
- ✓ Root cause analysis

Fix

- ✓ Data cleaning

Optimize

- ✓ Performance analytics
- ✓ Code analytics