Statistical Debugging for Real-World Performance Problems

Linhai Song[1] and Shan Lu[2]

[1]University of Wisconsin-Madison

[2]University of Chicago

What are Performance Problems?

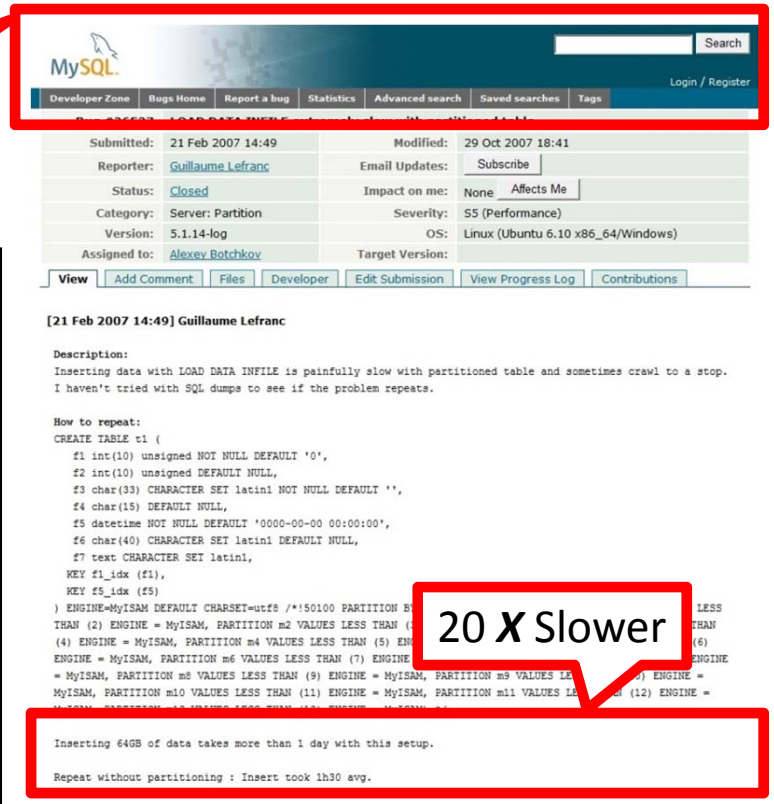
- Definition of Performance Problems (PPs):
  - Implementation mistakes causing inefficiency
- An example

rows=***0*** causing
no cache allocated

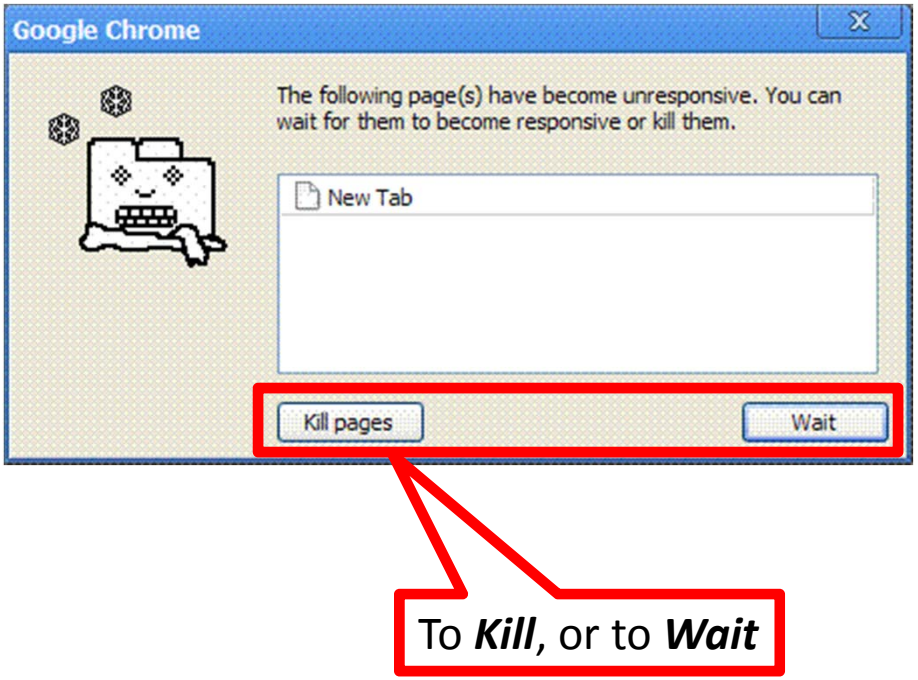
MySQL Bug DB

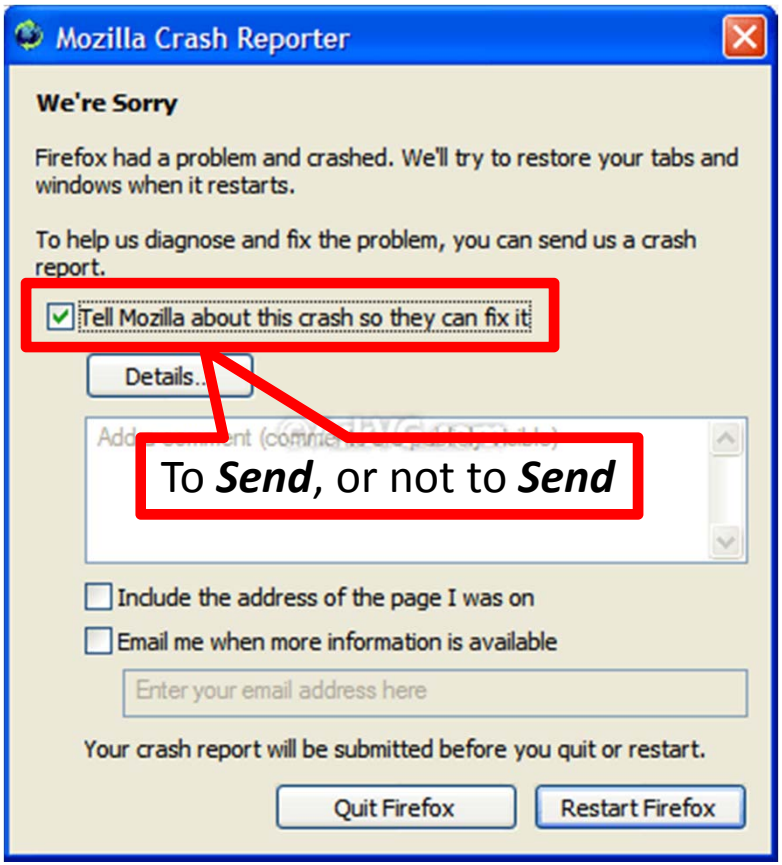
```
void ha_partition::start_bulk_insert(int rows) {
      .......
-     if (!rows)  //check whether rows is 0
-       DBUG_VOID_RETURN;
-     rows= rows/m_tot_parts + 1;
+     rows= rows ? rows/m_tot_parts + 1 : 0;
      ....... // fast path using caches
}
```

MySQL Bug 26527



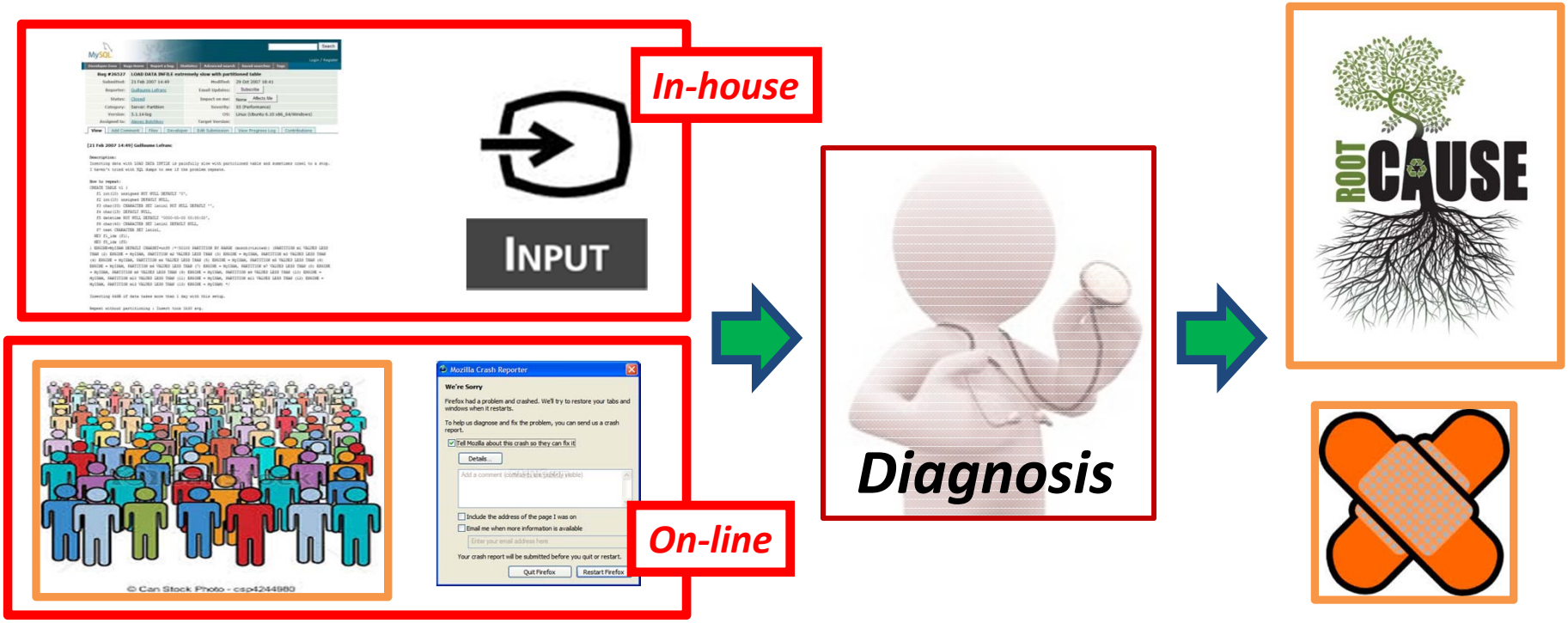
Performance Diagnosis





Performance Diagnosis

- Identifying the causes of performance failures
  - In-house and on-line diagnosis



Performance Diagnosis is Challenging

- The state of the art is ***preliminary***
  - Profilers
    - Only tools mentioned in bug reports we studied
    - Output time-consuming functions, not root causes
- More effective tools are necessary

```
void ha_partition :start_bulk_insert(int rows) {
        .......
-       if (!rows)
-           DBUG_VOID_RETURN;
-       rows= rows/m_tot_parts + 1;
+       rows= rows ? rows/m_tot_parts + 1 : 0;
        ....... // fast path using caches
}
```

Not in profiling results

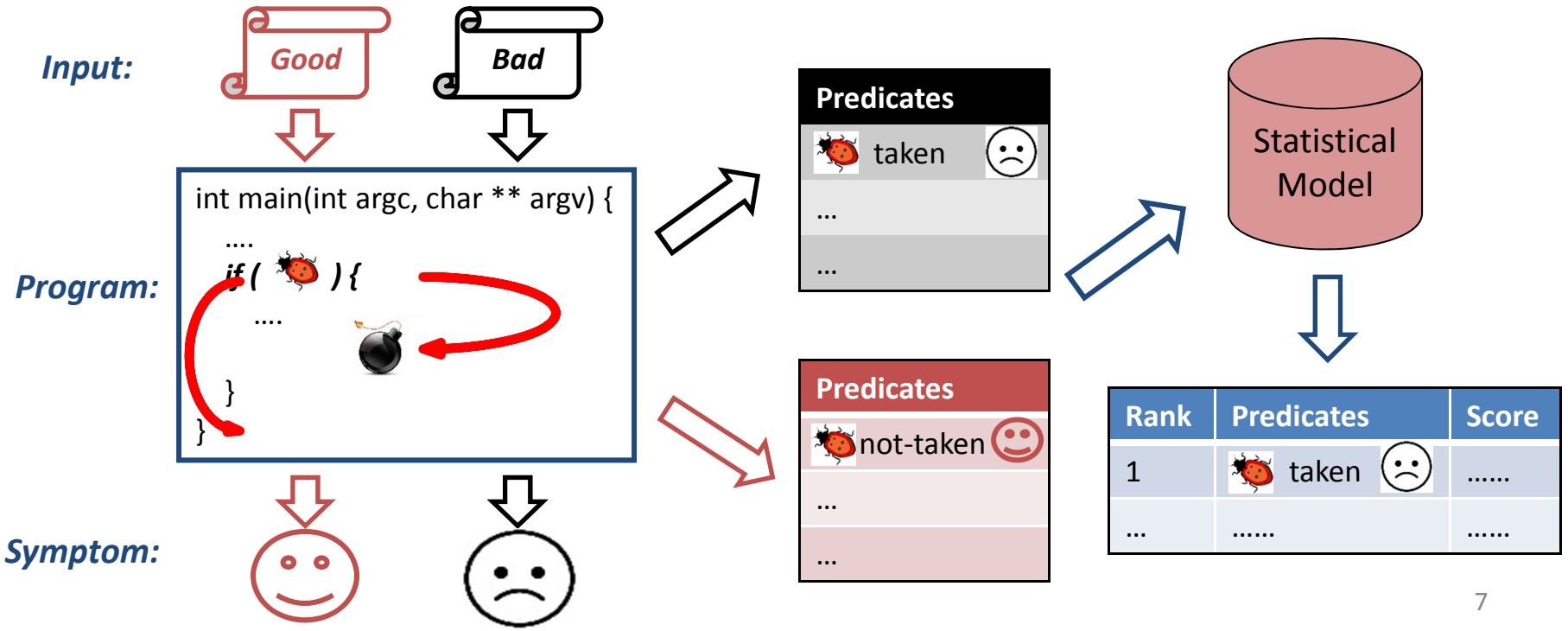
MySQL Bug 26527

How Can We Do Better?

Can we learn from functional bug diagnosis?

How to Diagnose Functional Bugs

- The state of the art is ***mature***
- Statistical Debugging(SD)
  - Among the most effective



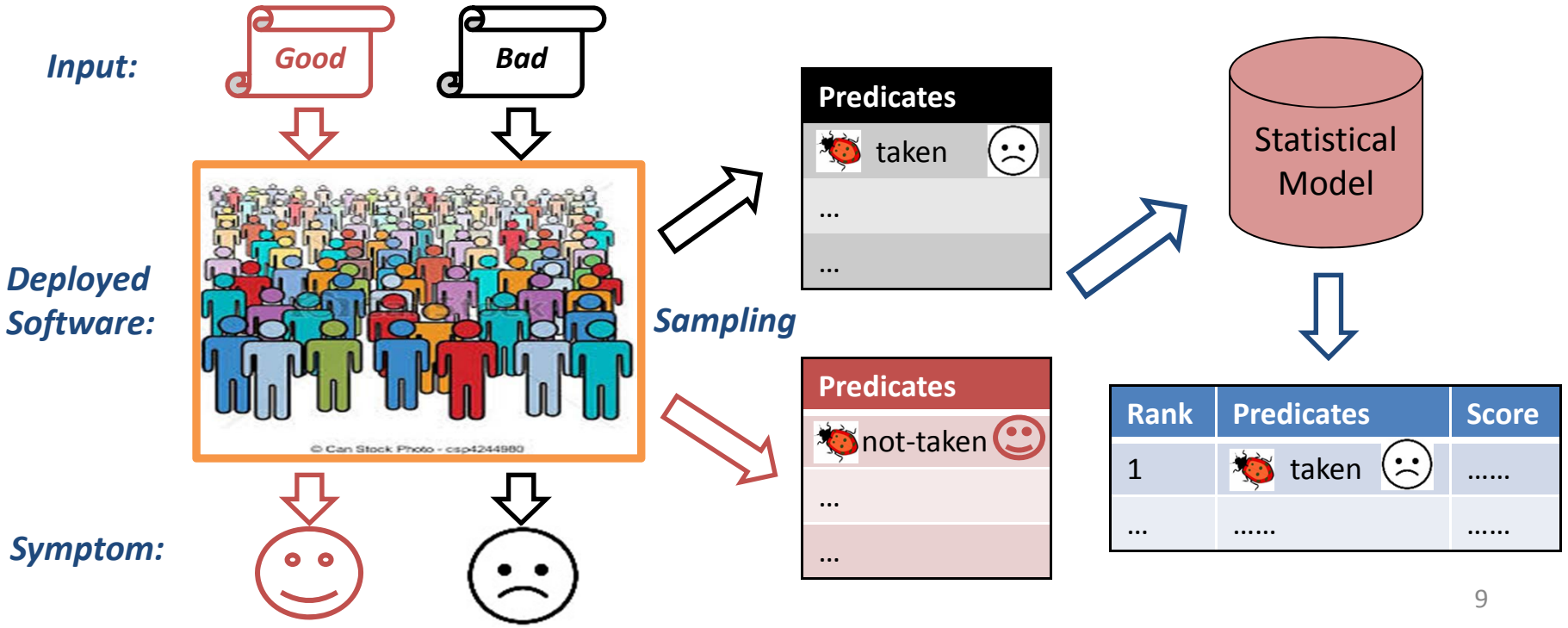
Remember these examples?

```
152  void
153  more_arrays ()
154  {
155    int indx;
156    int old_count;
157    bc_var_array **old_ary;
158    char **old_names;
159
160    /* Save the old values. */
161    old_count = a_count;
162    old_ary = arrays;
163    old_names = a_names;
164
165    /* Increment by a fixed amount and allocate. */
166    a_count += STORE_INCR;
167    arrays = (bc_var_array **) bc_malloc (a_count*si...
168    a_names = (char **) bc_malloc (a_count*sizeof(ch...
169
170    /* Copy the old arrays. */
171    for (indx = 1; indx < old_count; indx++)
172      arrays[indx] = old_ary[indx];
173
174
175    /* Initialize the new elements. */
176    for (; indx < v_count; indx++)
177      arrays[indx] = NULL;
178
179    /* Free the old elements. */
180    if (old_count != 0)
181      {
182        free (old_ary);
183        free (old_names);
184      }
185  }
```

```
1  // Print_tokens2 v7
2  if(ch == '\n')
3     return (TRUE);
4  else if(ch == ' ')
5  // Bug: should return FALSE
6     return (TRUE);
7  else
8     return (FALSE);
```

How to Diagnose Functional Bugs

- The state of the art is ***mature***
- Statistical Debugging(SD)
  - Among the most effective



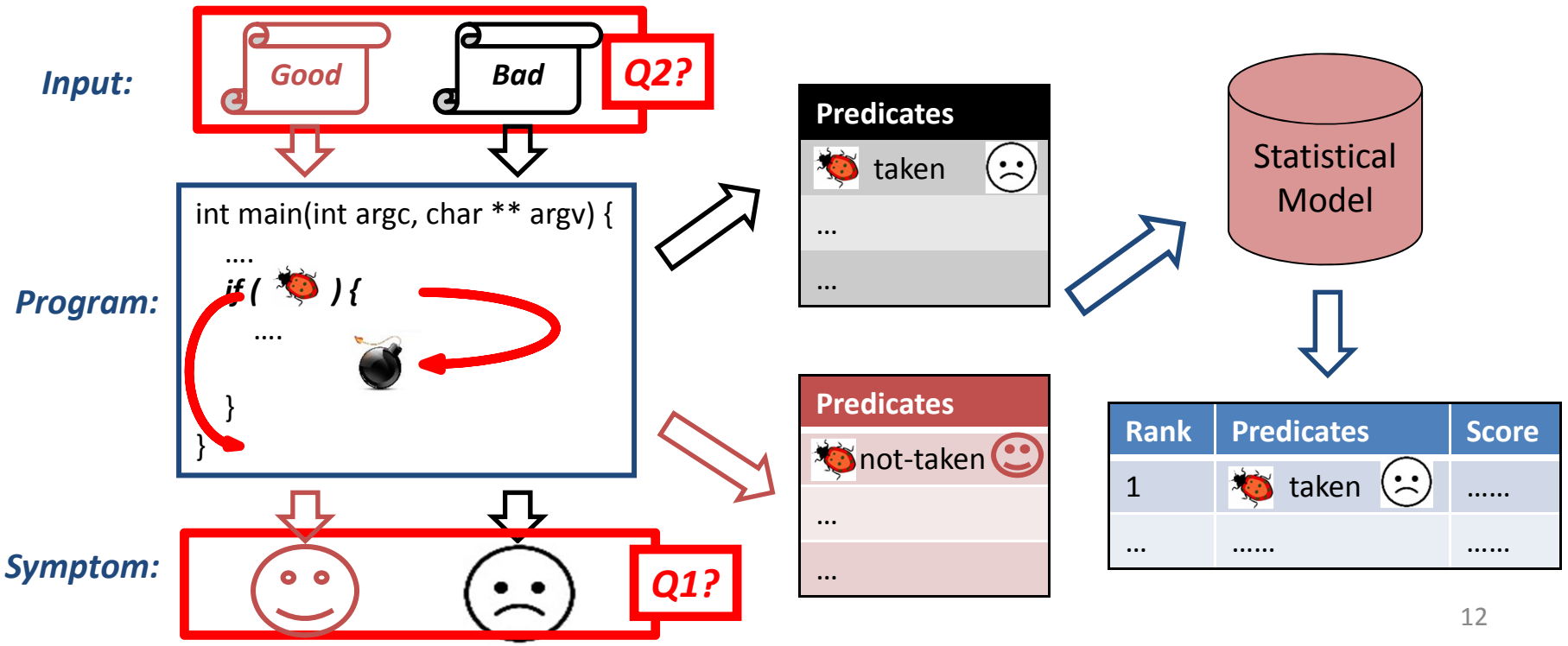
Apply SD to Performance Diagnosis?

Since statistical debugging is effective for functional diagnosis, maybe it will also be effective for performance diagnosis.

What are the challenges?

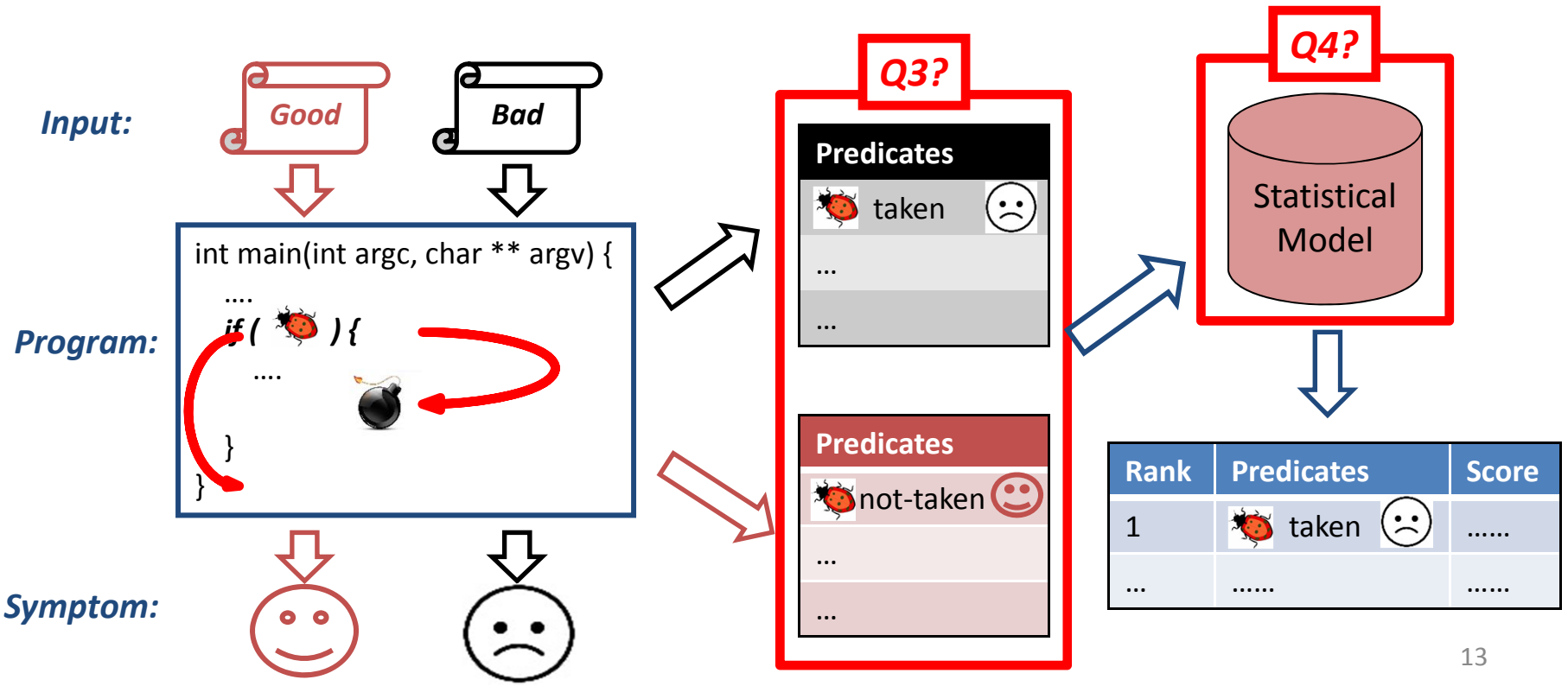
Is it Feasible?

- Q1: How to tell success runs from failure runs?
- Q2: How to obtain good and bad inputs?



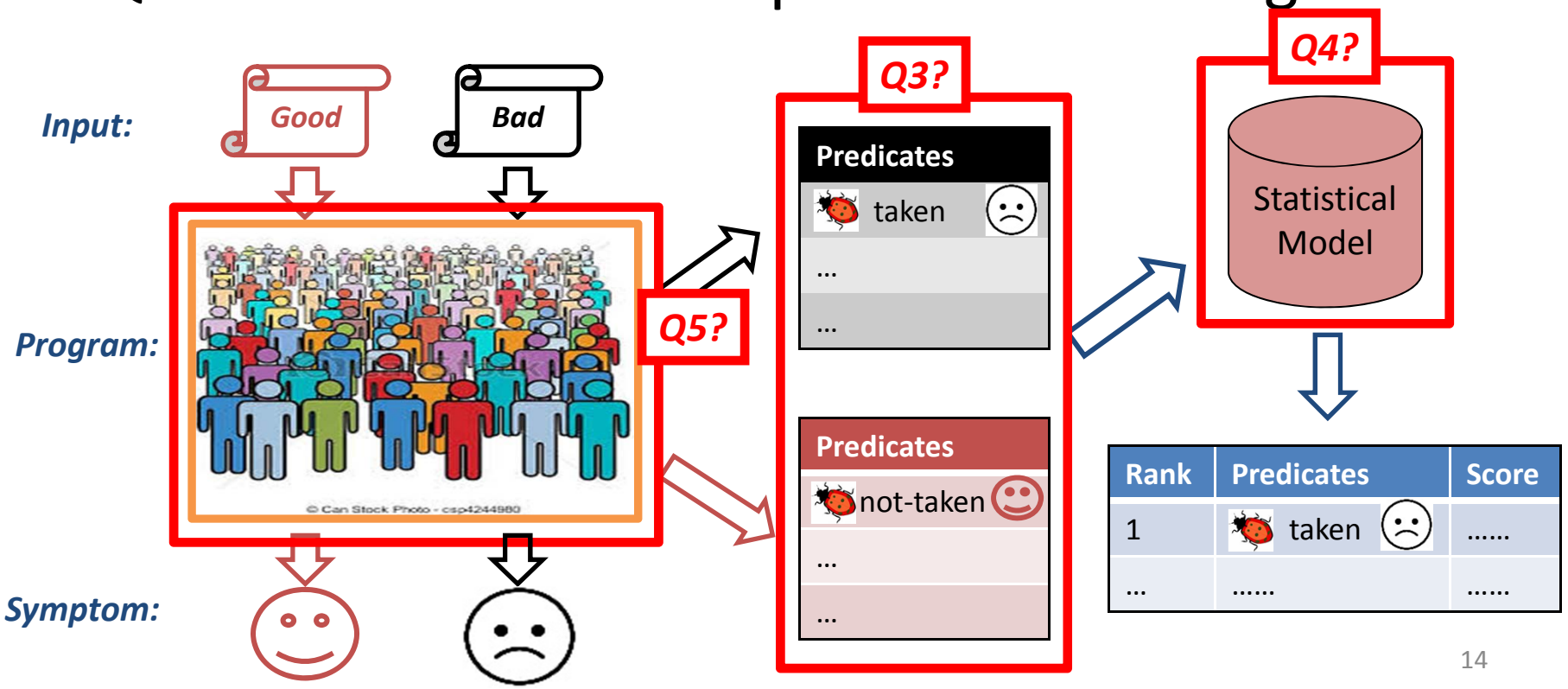
How to Apply?

- Q3: What predicates to collect?
- Q4: What statistical model to use?



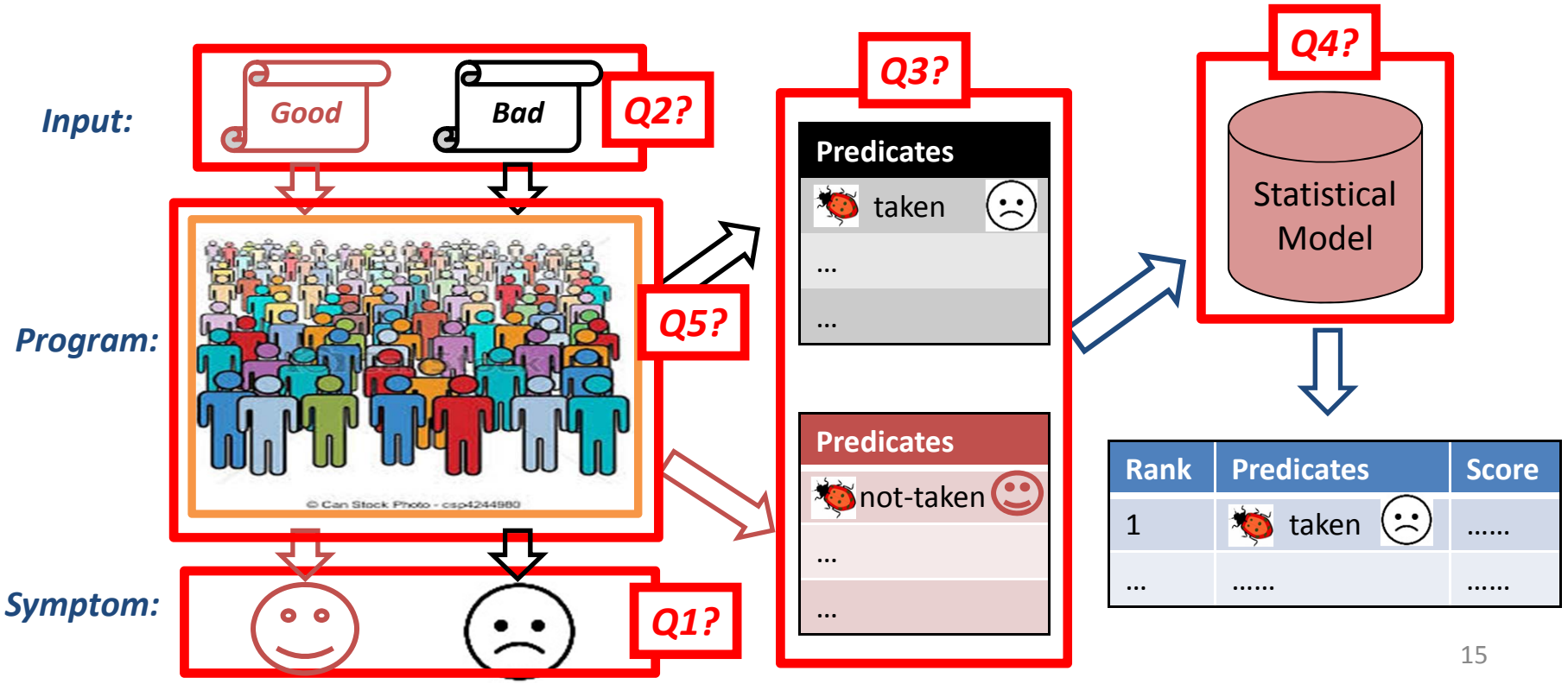
How to Apply?

- Q3: What predicates to collect?
- Q4: What statistical model to use?
- Q5: How to do on-line performance diagnosis?



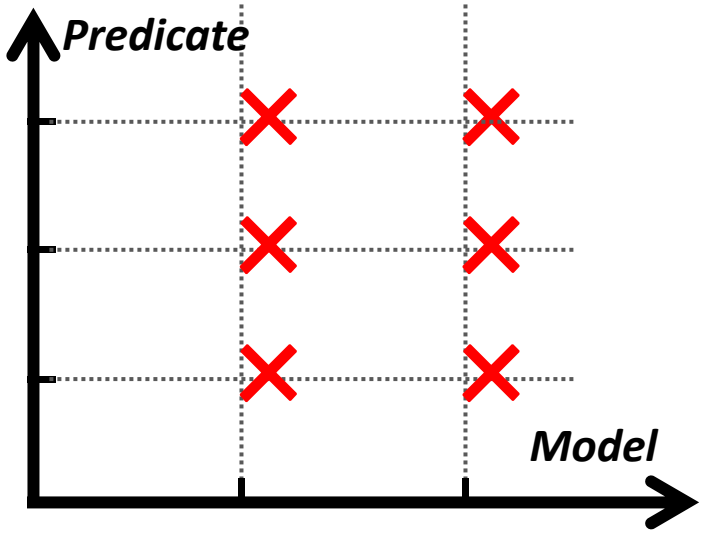
Contributions (I)

We answer all these questions by studying 65 user-reported performance bugs.



Contributions (II)

- Is it feasible to apply SD to PPs?
  - Easy to tell failure runs based on users' reports (Q1)
  - Inputs are provided during reporting (Q2)
- How to apply SD to PPs?
  - In-house diagnosis
    - 3 predicates (Q3)
    - 2 statistical models (Q4)
  - On-line diagnosis (Q5)
    - Same diagnosis capability with <10% overhead
    - Not sacrifice diagnosis latency (**Unique**)



Outline

- Overview
- Is it feasible to apply SD for PPs?
- How to conduct SD for PPs?
  - In-house diagnosis scenario
  - On-line diagnosis scenario
- Conclusion

Methodology

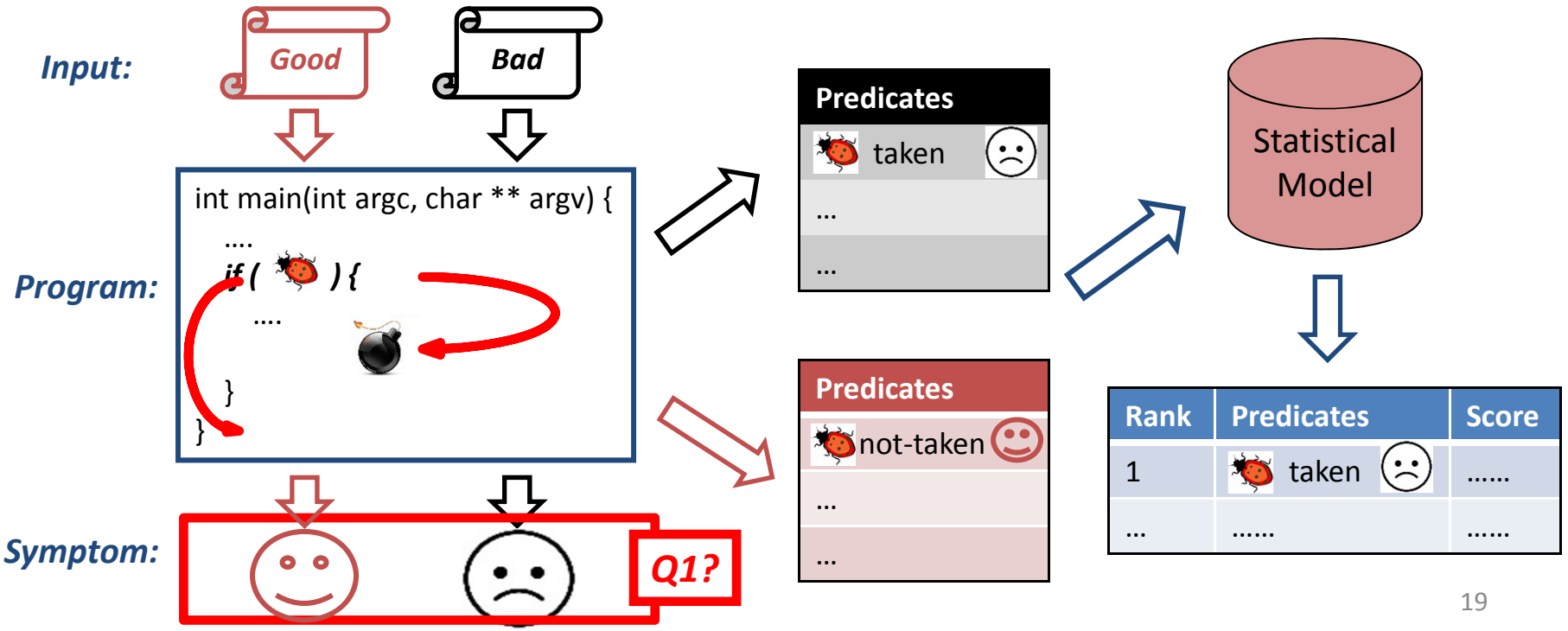
- Application and Bug Source

App.	# Bugs[1]	# Bug User Perceived
Apache	25	16
Chrome	10	5
GCC	11	9
Mozilla	36	19
MySQL	28	17
Total: 110		65

[1] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and Detecting Real-World Performance Bugs. In PLDI'2012.

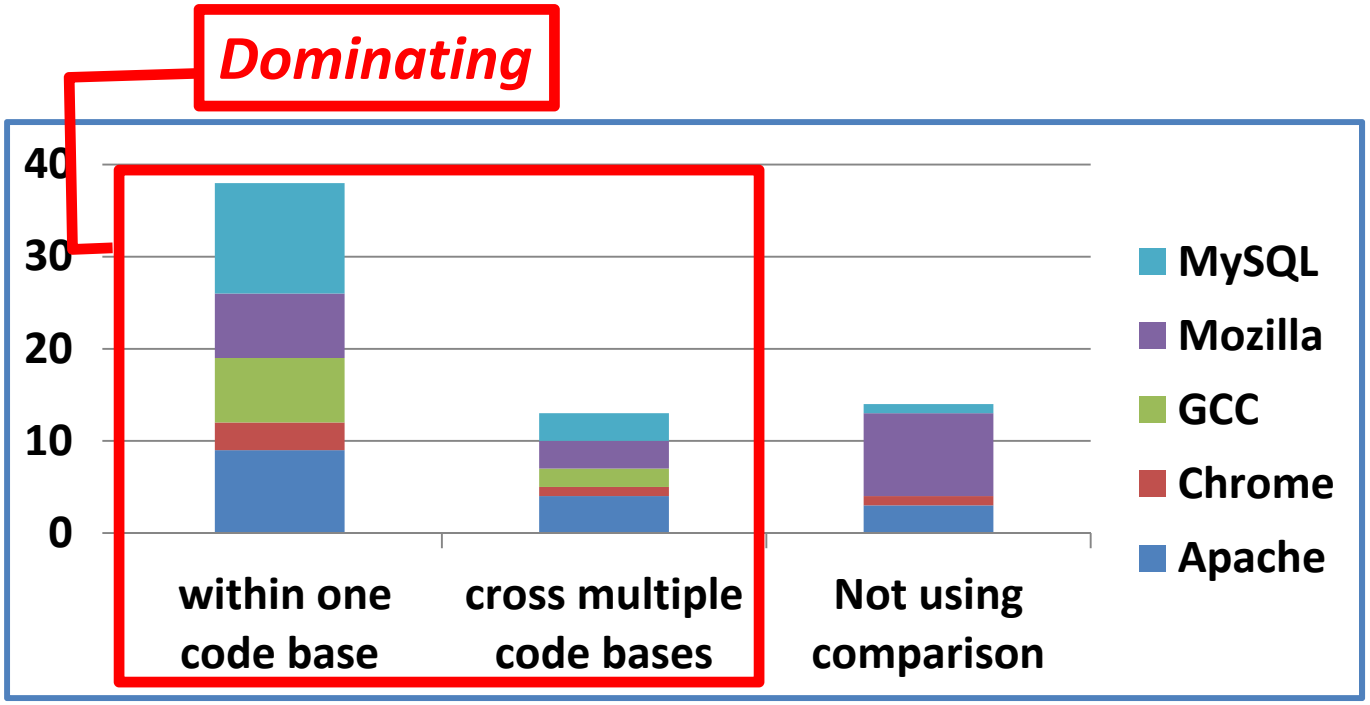
Is It Feasible? (Part I)

- **Q1: How to tell success runs from failure runs?**
  - A large workload? Or inefficient implementation?



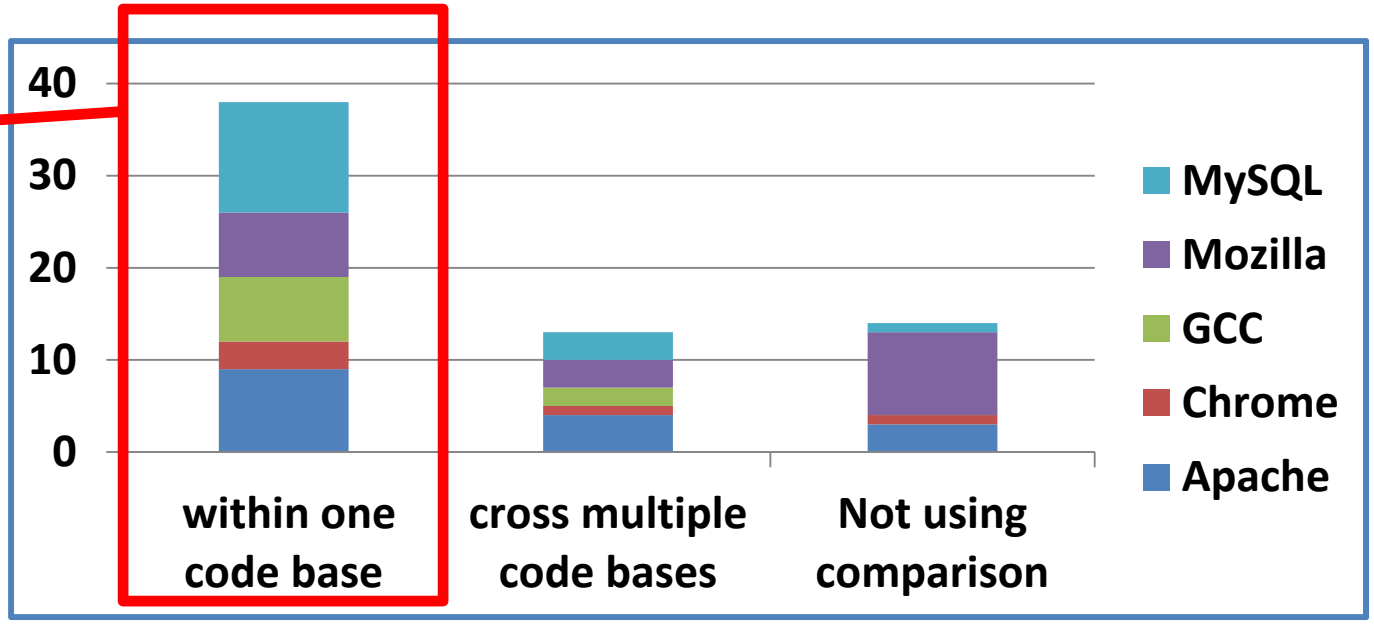
Q1: How to Identify Failure Runs?

- The majority is observed through comparison



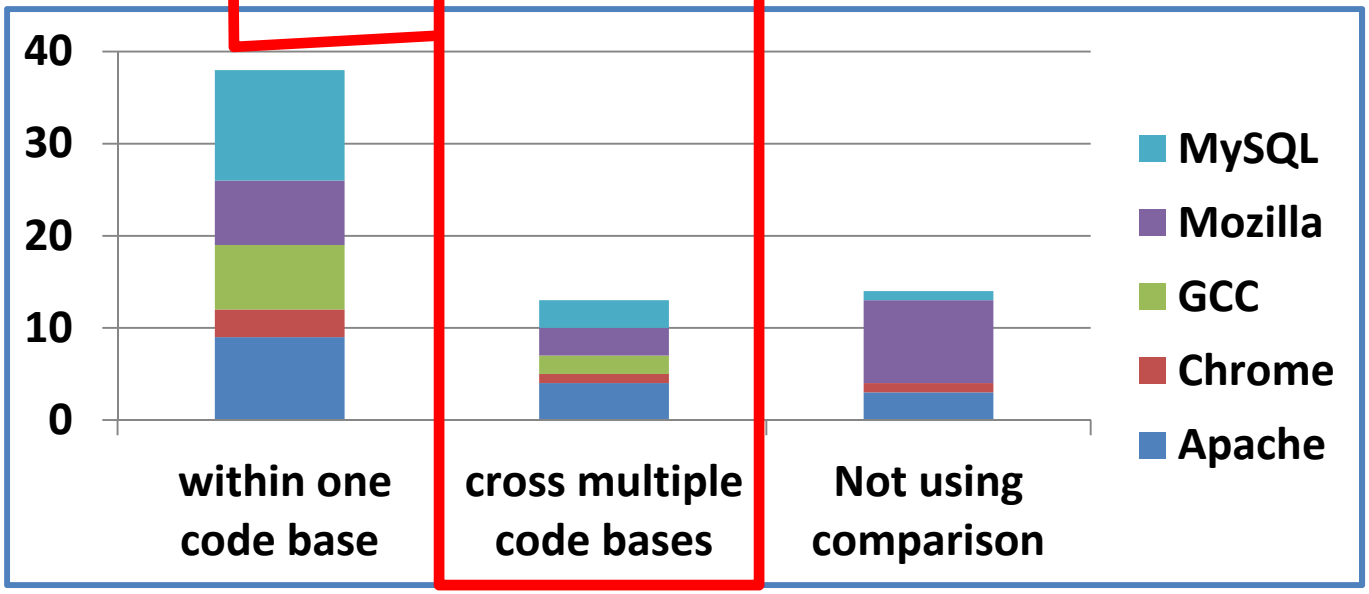
Comparison within One Code Base

- the same input with different ***configuration***
- inputs with different ***sizes***
- inputs with slightly different ***functionality***



Comparison across Multiple Code Bases

- same applications' different ***versions***
- different ***applications***

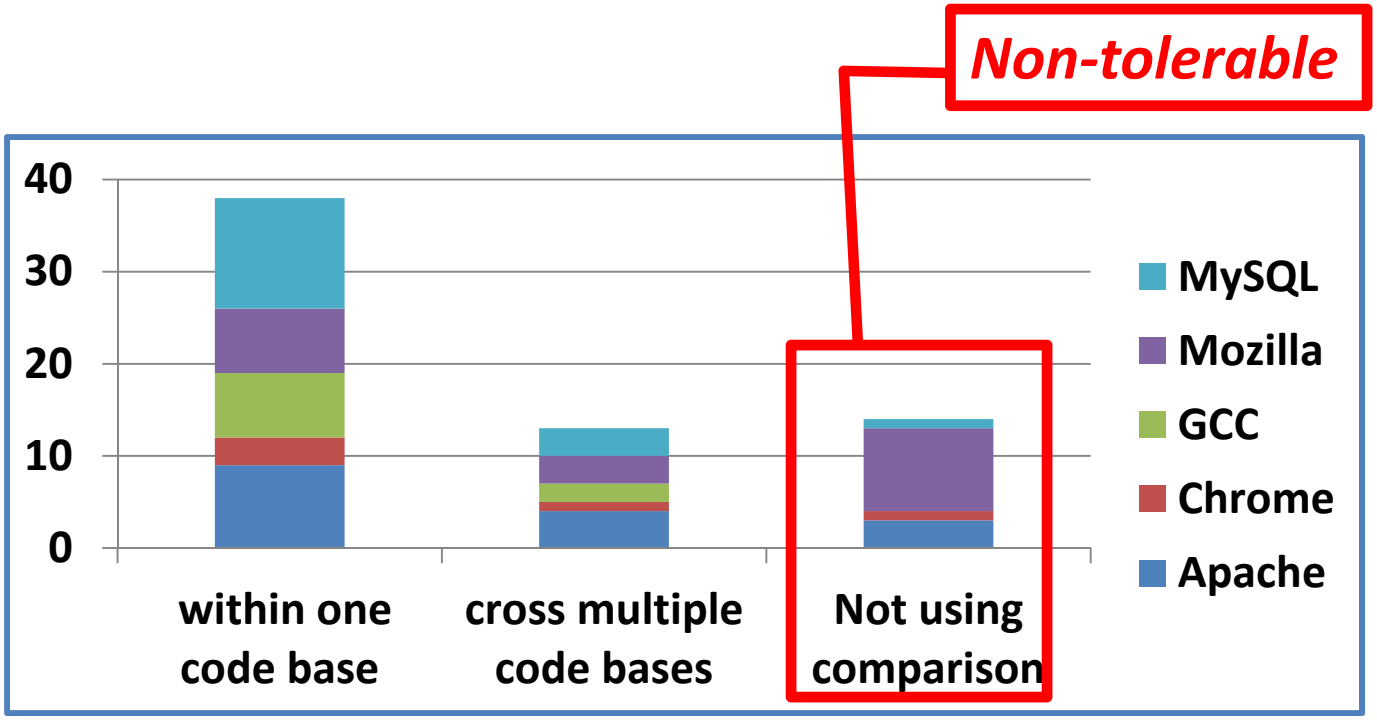


Not Using Comparison

Mozilla#299742: "it frozen the GUI to crawl"

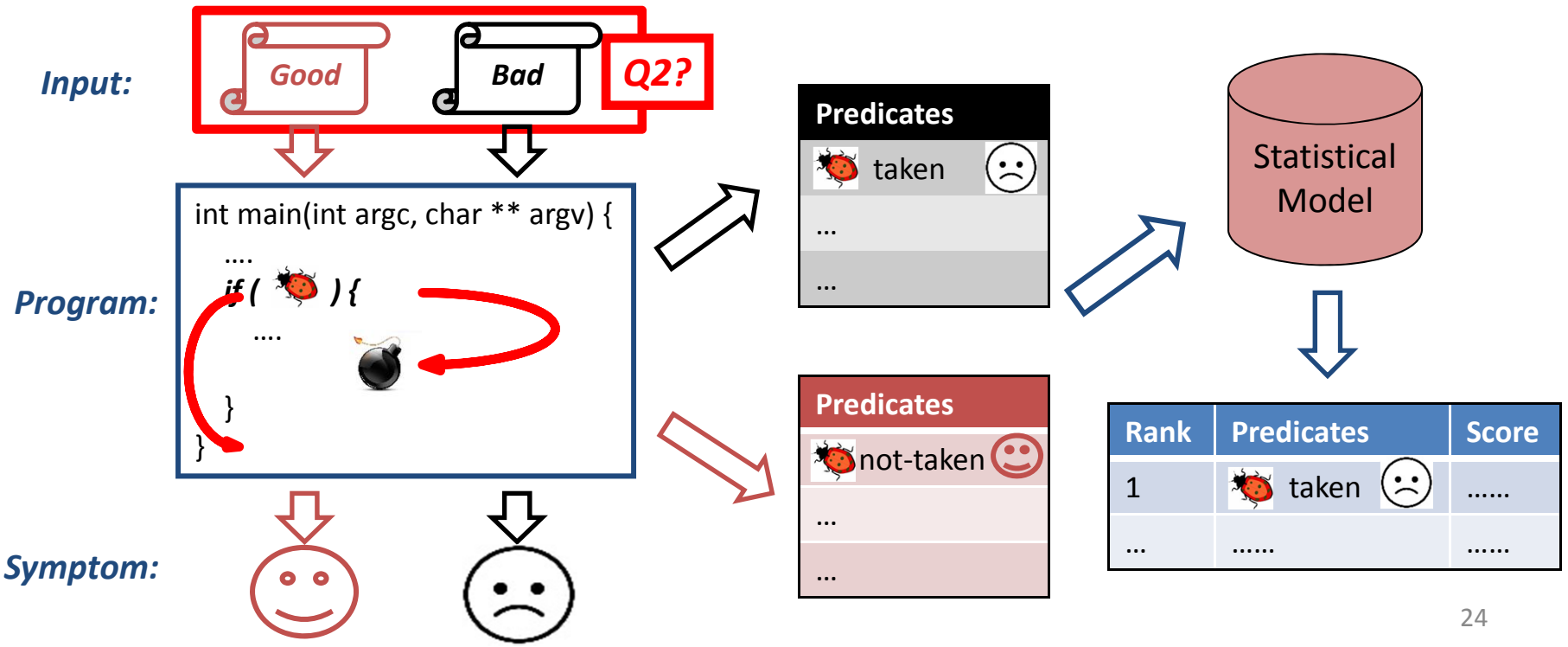
Mozi ***Easy to tell failures from successes!*** h the page"

MySQL46461: "causing the test suit to fail due to timeout"



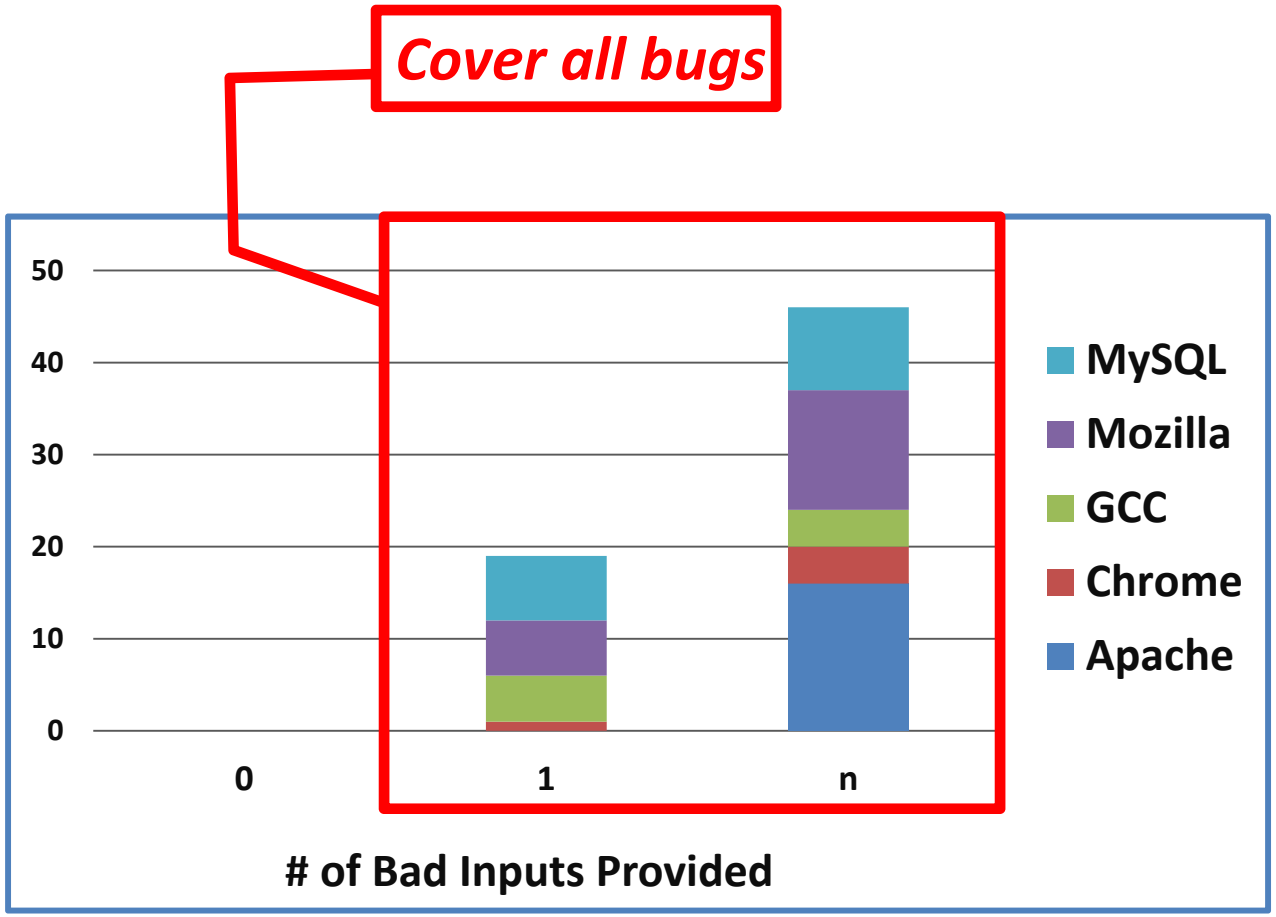
Is It Feasible? (Part II)

- Q1: How to tell success runs from failure runs?
- **Q2: How to obtain good and bad inputs?**



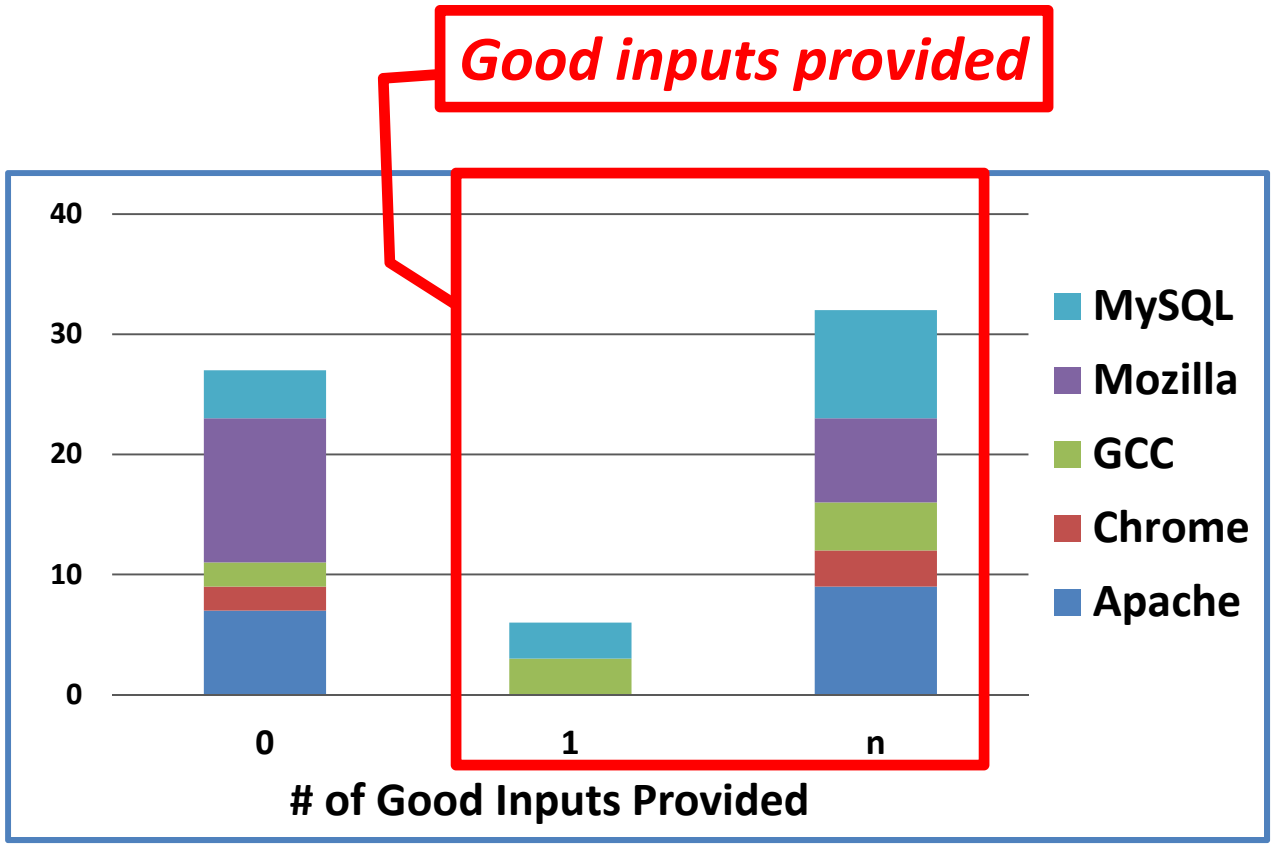
How to Obtain Bad Inputs?

- Bad inputs are provided in all bug reports

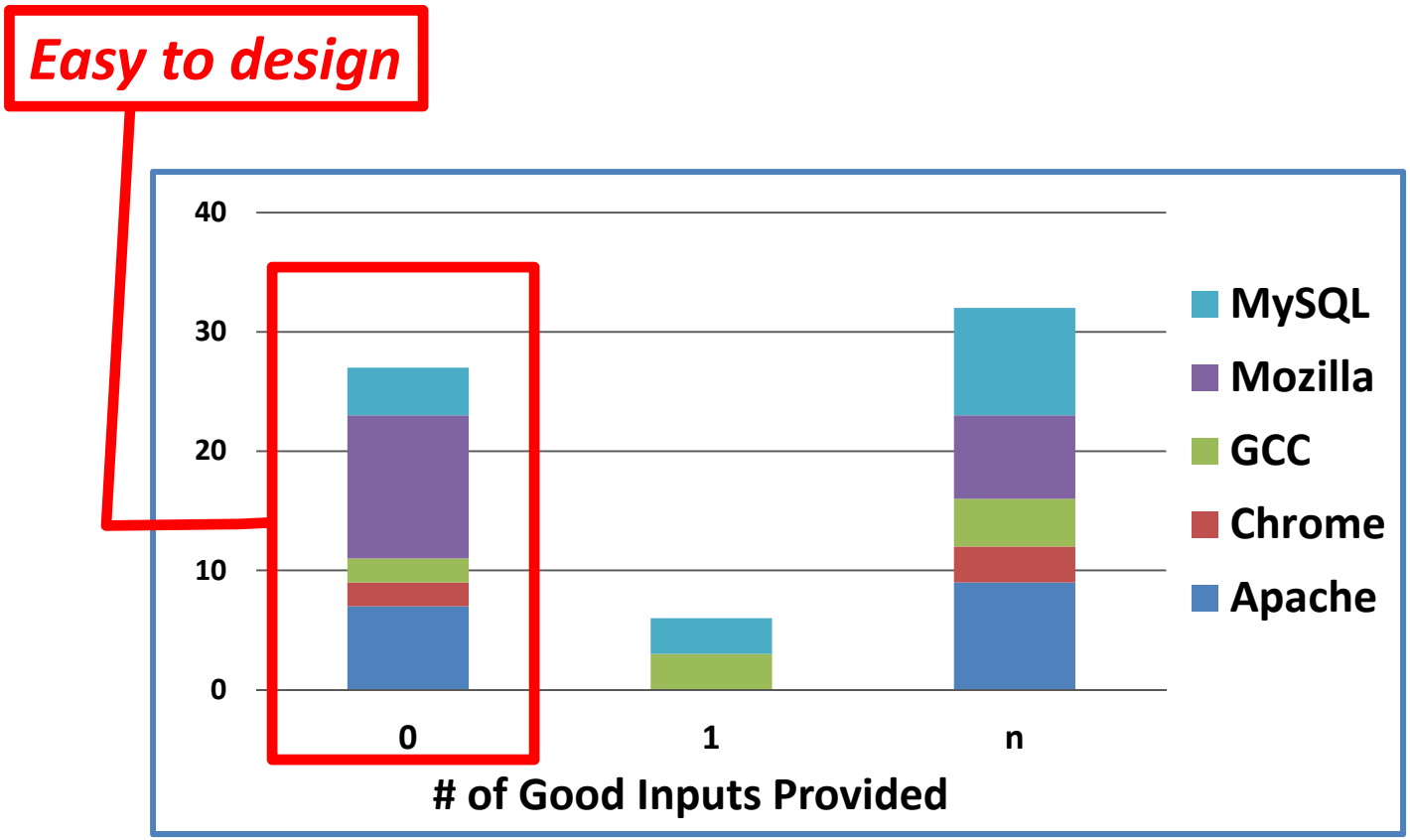


How to Obtain Good Inputs? (I)

- 60% contain good inputs



How to Obtain Good Inputs? (II)



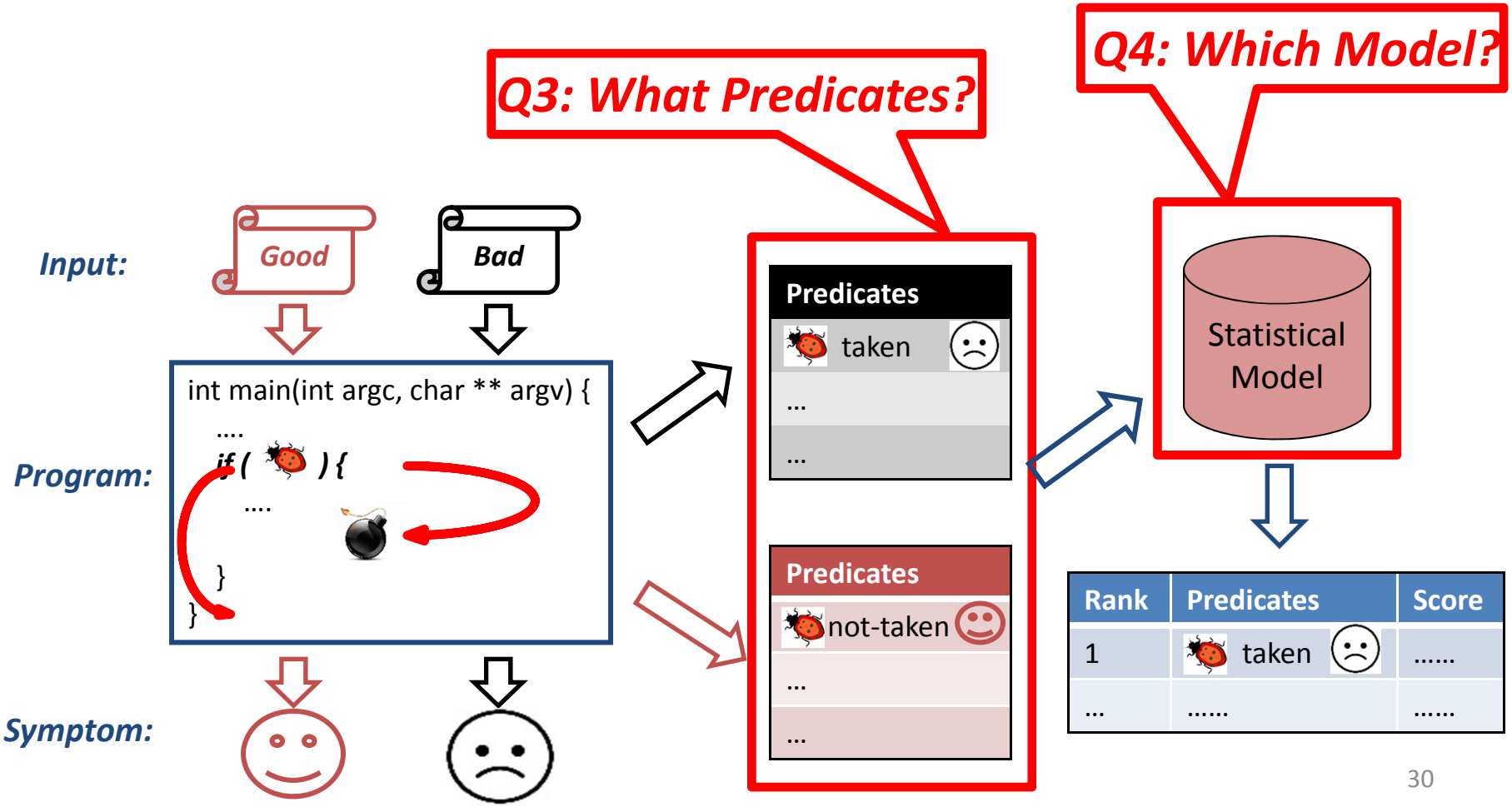
Other Findings and Implications

- Compared with functional bugs
  - More PPs observed through comparison
  - More PPs reported with good inputs
- Implications for SD
  - Easy to tell success runs from failure runs
  - Similar good inputs are provided
  - SD is a nature fit for PPs

Outline

- Overview
- Is it feasible to apply SD for PPs?
- How to conduct SD for PPs?
  - In-house diagnosis scenario
  - On-line diagnosis scenario
- Conclusion

In-house Statistical Debugging



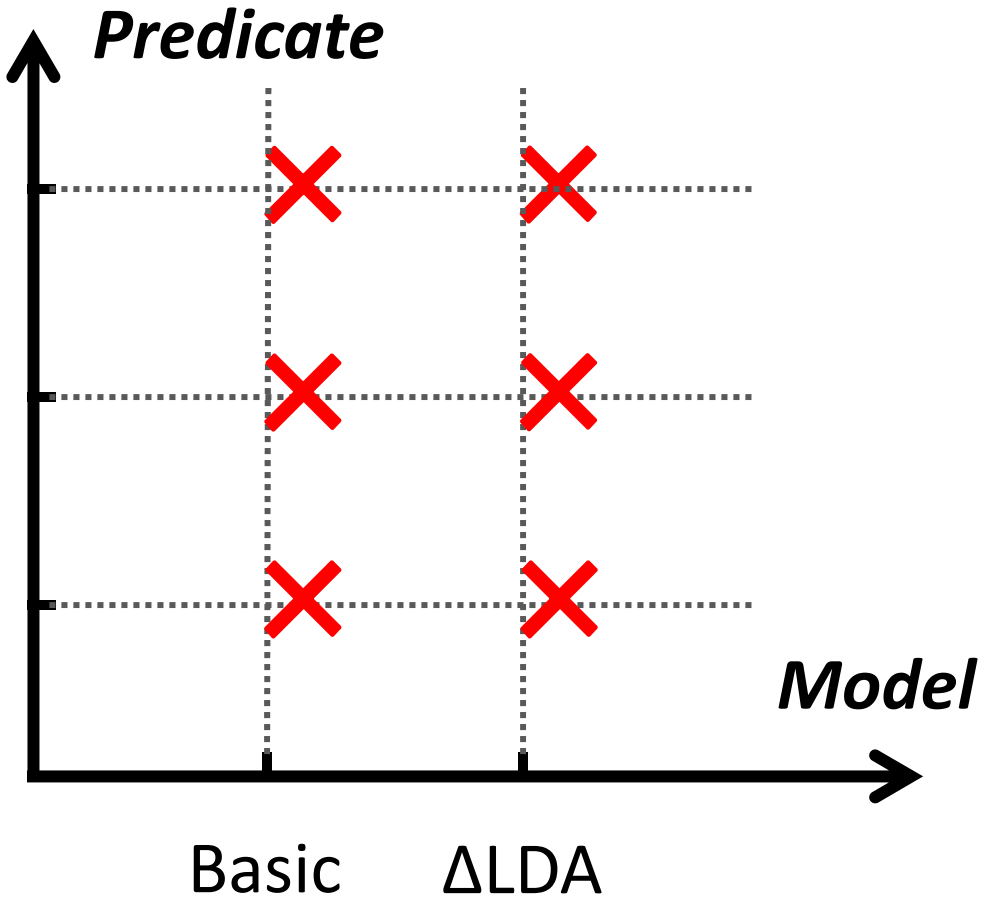
Design

- Study widely used predicates and models

```
int x, y;
…
x = …;
```

```
n=fprintf(…);
```

```
if (p)  …
else  ….
```



Experimental Methodology

- Experimental setting
  - 10 success runs vs. 10 failure runs
  - 20 unique inputs
- Techniques under comparison
  - CBI[2, 3] for C programs
  - Pin for C++ programs
  - Compared with profiling results from OProfile

[2] Ben Liblit, Alex Aiken, Alice X Zhen, and Michael I Jordan. Bug Isolation via Remote Program Sampling. In PLDI'2003.
[3] Ben Liblit, Mayur Naik, Alice X Zheng, Alex Aiken, and Michael I Jordan. Scalable Statistical Bug Isolation. In PLDI'2005.

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	**Branch**	**Return**	**S-pair**	**Branch**	**Return**	**S-pair**	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	**Branch**	**Return**	**S-pair**	Branch	Return	S-pair	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-			-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Most Useful

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	Branch	Return	S-pair	**Branch**	**Return**	**S-pair**	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	**Branch**	Return	S-pair	**Branch**	**Return**	**S-pair**	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	Branch	Return	S-pair	Branch	Return	S-pair	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Experimental Results

	Basic Model			ΔLDA			Profiler
BugID	**Branch**	Return	S-pair	Branch	Return	S-pair	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Experimental Results

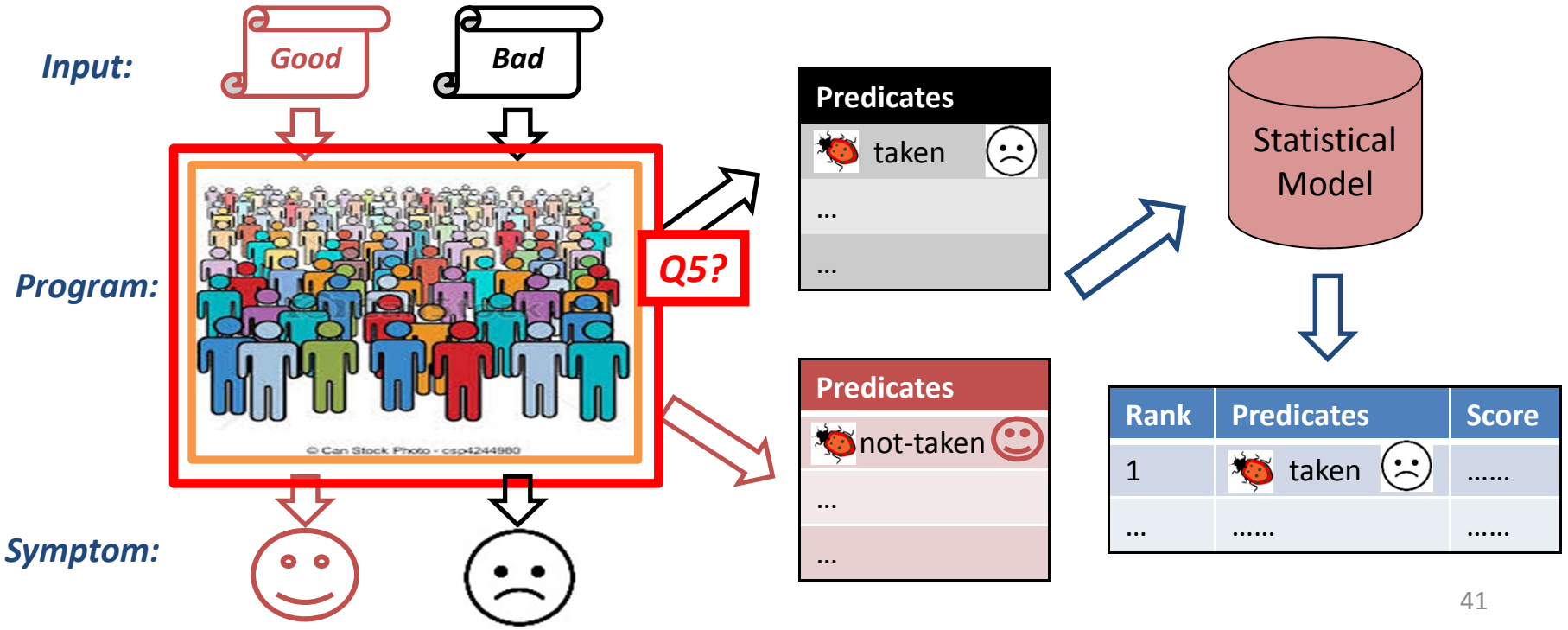
	Basic Model				ΔLDA		Profiler
BugID	Branch	Return	S-pair	**Branch**	Return	S-pair	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
MySQL42649	√1	-	/	-	-	/	-
MySQL44723	√1	-	/	-	-	/	-
Apache3278	√1	√1	√1	-	-	-	-
Apache34464	-	-	-	√3	√1	-	√5
…	…	…	…	…	…	…	…

Outline

- Overview
- Is it feasible to apply SD for PPs?
- How to conduct SD for PPs?
  - In-house diagnosis scenario
  - On-line diagnosis scenario
- Conclusion

On-line Statistical Debugging

- Q5: How to do on-line performance diagnosis?
  - Less information from one single run
  - Diagnosis capability relies on multiple runs info.



Experimental Methodology

- Tool design
  - CBI in sampling mode for C benchmarks
  - LBR for C++ benchmarks
- Benchmarks
  - Reuse benchmarks from in-house experiments
  - Most effective predicate and model
- Experiment design
  - Default sampling rate is roughly 1/10000
  - 1000 success runs and 1000 failure runs

Experimental Results

BugID	Diagnosis Capability	Run-time Overhead	Requested Failure Runs
Mozilla258793	√1	2.39%	100
Mozilla299742	√1	4.27%	500
Mozilla347306	√1	1.42%	10
Mozilla416628	√1	2.03%	10
MySQL15811	√1	2.25%	10
MySQL26527	√1	6.05%	500
MySQL27287	√1	3.02%	10
MySQL40337	√1	2.69%	100
MySQL42649	√2	6.10%	500
MySQL44723	√1	3.16%	100
Apache3278	-	0.23%	>1000
Apache34464	√3	0.18%	10
…	…	…	…

Experimental Results

BugID	Diagnosis Capability	Run-time Overhead	Requested Failure Runs
	√1	2.39%	
	√1	4.27%	
	√1	1.42%	10
Mozilla416628	√1	2.03%	10
MySQL15811	√1	2.25%	10
MySQL26527	√1	6.05%	500
MySQL27287	√1	3.02%	10
MySQL40337	√1	2.69%	100
MySQL42649	√2	6.10%	500
MySQL44723	√1	3.16%	100
Apache3278	-	0.23%	>1000
Apache34464	√3	0.18%	10
…	…	…	…

Same Diagnosis Capability

< 10%

Experimental Results

BugID	Diagnosis Capability	Run-time Overhead	Requested Failure Runs
Mozilla258793	v1	2.39%	
Mozilla299742	v1	4.27%	
Mozilla347306	v1	1.42%	
Mozilla416628	v		
MySQL15811	v1	2.25%	
MySQL26527	v1	6.05%	
MySQL27287	v1	3.02%	
MySQL40337	v1	2.69%	
MySQL42649	v2	6.10%	
MySQL44723	v1	3.16%	
Apache3278	-	0.23%	
Apache34464	v3	0.18%	
…	…	…	…

10 X 10000 ??

Experimental Results

BugID	Diagnosis Capability	Run-time Overhead	Requested Failure Runs
Mozilla258793	V1	2.39%	100
Mozilla299742	V1	4.27%	500
Mozilla347306	V1	1.42%	10
Mozilla416628	V1	2.03%	10
MySQL15811	V1	2.25%	10
MySQL26527	V1	6.05%	500
MySQL27287	V1	3.02%	10
MySQL40337	V1	2.69%	100
MySQL42649	V2	6.10%	500
MySQL44723	V1	3.16%	100
Apache3278	-	0.23%	>1000
Apache34464	V3	0.18%	10
…	…	…	…

Experimental Results

BugID	Diagnosis Capability	Run-time Overhead	Requested Failure Runs
Mozilla258793	√1	2.39%	100
Mozilla299742	√1	4.27%	500
Mozilla347306	√1	1.42%	10
Mozilla416628	√1	2.03%	10
MySQL15811	√1	2.25%	10
MySQL26527	√1	6.05%	500
MySQL27287	√1	3.02%	10
MySQL40337	√1	2.69%	100
MySQL42649	√2	6.10%	500
MySQL44723	√1	3.16%	100
Apache3278	-	0.23%	>1000
Apache34464	√3	0.18%	10
…	…	…	…

Why?

Conclusion and Future Works

- Study diagnosis process for PPs
  - Statistical debugging is a natural fit
- Study statistical debugging on PPs
  - Branch predicates + two statistical models
- Future works
  - Analyze inefficient loops
  - Provide detailed fix hints

Break