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?

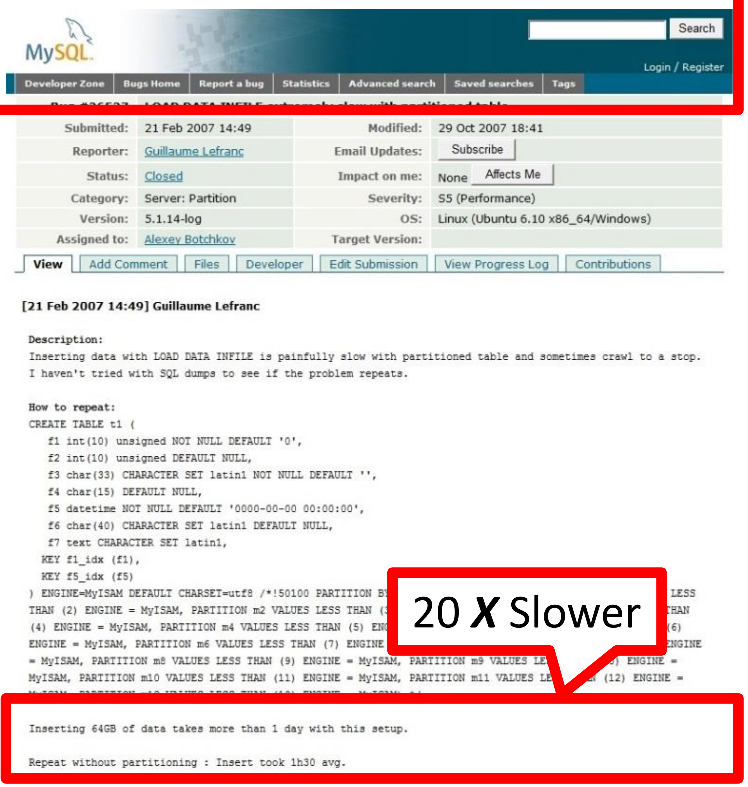
- Definition of Performance Problems (PPs):
  - Implementation mistakes causing inefficiency
- An example

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



Submitted:	21 Feb 2007 14:49	Modified:	29 Oct 2007 18:41
Reporter:	Guillaume Lefranc	Email Updates:	Subscribe
Status:	Closed	Impact on me:	None Affects Me
Category:	Server: Partition	Severity:	S5 (Performance)
Version:	5.1.14-log	OS:	Linux (Ubuntu 6.10 x86_64/Windows)
Assigned to:	Alexey Botchkov	Target Version:	

[21 Feb 2007 14:49] Guillaume Lefranc

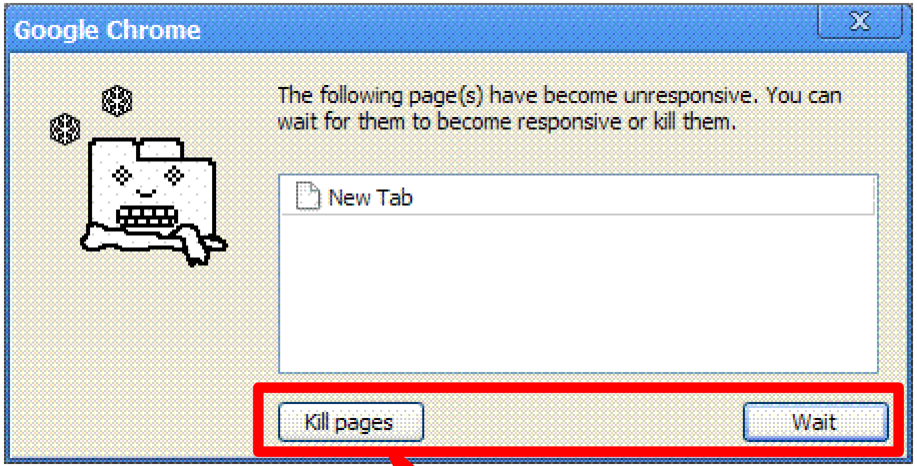
```
Description:
Inserting data with LOAD DATA INFILE is painfully slow with partitioned table and sometimes crawl to a stop.
I haven't tried with SQL dumps to see if the problem repeats.

How to repeat:
CREATE TABLE t1 (
   f1 int(10) unsigned NOT NULL DEFAULT '0',
   f2 int(10) unsigned DEFAULT NULL,
   f3 char(33) CHARACTER SET latin1 NOT NULL DEFAULT '',
   f4 char(15) DEFAULT NULL,
   f5 datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
   f6 char(40) CHARACTER SET latin1 DEFAULT NULL,
   f7 text CHARACTER SET latin1,
  KEY f1_idx (f1),
  KEY f5_idx (f5)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 /*!50100 PARTITION BY                        LESS
THAN (2) ENGINE = MyISAM, PARTITION m2 VALUES LESS THAN (                         THAN
(4) ENGINE = MyISAM, PARTITION m4 VALUES LESS THAN (5) ENG                        (6)
ENGINE = MyISAM, PARTITION m6 VALUES LESS THAN (7) ENGINE                         ENGINE
= MyISAM, PARTITION m8 VALUES LESS THAN (9) ENGINE = MyISAM, PARTITION m9 VALUES LE      ) ENGINE =
MyISAM, PARTITION m10 VALUES LESS THAN (11) ENGINE = MyISAM, PARTITION m11 VALUES LE     (12) ENGINE =

Inserting 64GB of data takes more than 1 day with this setup.

Repeat without partitioning : Insert took 1h30 avg.
```

20 ***X*** Slower

Fighting PPs is Important

- PPs widely exists in production run software
  - 5 to 60 Mozilla PPs are fixed each month
- PPs are getting more important
  – Hardware is not getting faster (per-core)
  – Software is getting more complex
  – Energy saving is getting more urgent

Performance Diagnosis

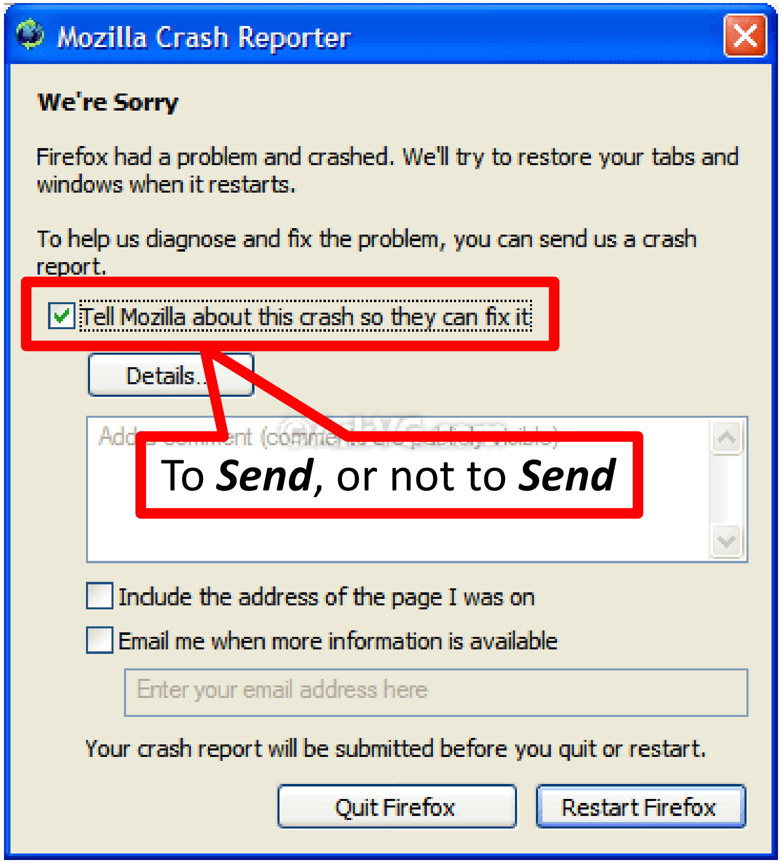


Google Chrome

The following page(s) have become unresponsive. You can wait for them to become responsive or kill them.

New Tab

To ***Kill***, or to ***Wait***



Mozilla Crash Reporter

We're Sorry

Firefox had a problem and crashed. We'll try to restore your tabs and windows when it restarts.

To help us diagnose and fix the problem, you can send us a crash report.

☑ Tell Mozilla about this crash so they can fix it

Add a comment (comments are publicly visible)

□ Include the address of the page I was on

□ Email me when more information is available

Enter your email address here

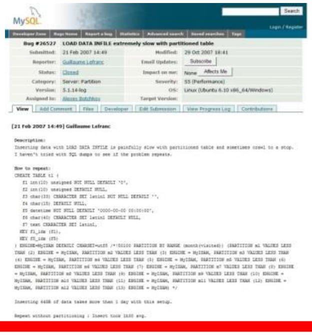
Your crash report will be submitted before you quit or restart.

To ***Send***, or not to ***Send***

Performance Diagnosis

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

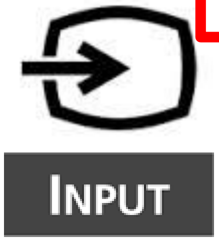
In-house

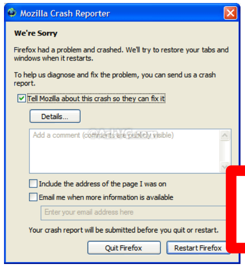


Bug #26527	LOAD DATA INFILE extremely slow with partitioned table		
Submitted:	21 Feb 2007 14:49	Modified:	29 Oct 2007 18:41
Reporter:	Guillaume Lefranc	Email Updates:	Subscribe
Status:	Closed	Impact on me:	None Affects Me
Category:	Server: Partition	Severity:	S5 (Performance)
Version:	5.1.14-log	OS:	Linux (Ubuntu 6.10 x86_64/Windows)
Assigned to:	Alexey Botchkov	Target Version:	

[21 Feb 2007 14:49] Guillaume Lefranc:

```
Description:
Inserting data with LOAD DATA INFILE is painfully slow with partitioned table and sometimes crawl to a stop.
I haven't tried with SQL dumps to see if the problem repeats.

How to repeat:
CREATE TABLE t1 (
   f1 int(10) unsigned NOT NULL DEFAULT '0',
   f2 int(10) unsigned DEFAULT NULL,
   f3 char(33) CHARACTER SET latin1 NOT NULL DEFAULT '',
   f4 char(15) DEFAULT NULL,
   f5 datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
   f6 char(40) CHARACTER SET latin1 DEFAULT NULL,
   f7 text CHARACTER SET latin1,
  KEY f1_idx (f1),
  KEY f5_idx (f5)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 /*!50100 PARTITION BY RANGE (month(visited)) (PARTITION m1 VALUES LESS
THAN (2) ENGINE = MyISAM, PARTITION m2 VALUES LESS THAN (3) ENGINE = MyISAM, PARTITION m3 VALUES LESS THAN
(4) ENGINE = MyISAM, PARTITION m4 VALUES LESS THAN (5) ENGINE = MyISAM, PARTITION m5 VALUES LESS THAN (6)
ENGINE = MyISAM, PARTITION m6 VALUES LESS THAN (7) ENGINE = MyISAM, PARTITION m7 VALUES LESS THAN (8) ENGINE
= MyISAM, PARTITION m8 VALUES LESS THAN (9) ENGINE = MyISAM, PARTITION m9 VALUES LESS THAN (10) ENGINE =
MyISAM, PARTITION m10 VALUES LESS THAN (11) ENGINE = MyISAM, PARTITION m11 VALUES LESS THAN (12) ENGINE =
MyISAM, PARTITION m12 VALUES LESS THAN (13) ENGINE = MyISAM) */

Inserting 44GB of data takes more than 1 day with this setup.

Repeat without partitioning : Insert took 1h30 avg.
```





We're Sorry

Firefox had a problem and crashed. We'll try to restore your tabs and windows when it restarts.

To help us diagnose and fix the problem, you can send us a crash report.

☑ Tell Mozilla about this crash so they can fix it

Details...
Add a comment (comments are publicly visible)

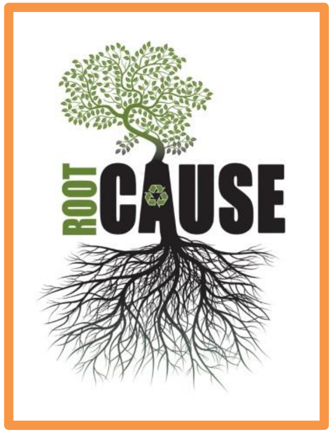
□ Include the address of the page I was on

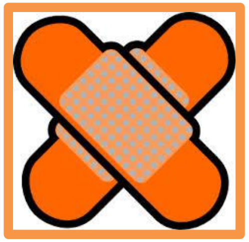
□ Email me when more information is available

Enter your email address here

Your crash report will be submitted before you quit or restart.

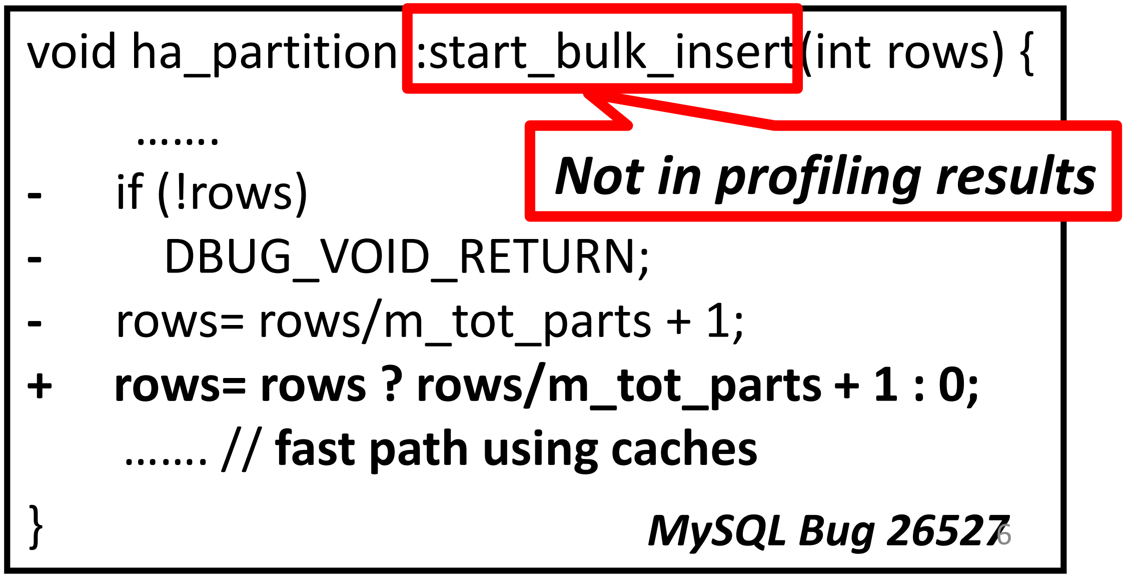
On-line





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

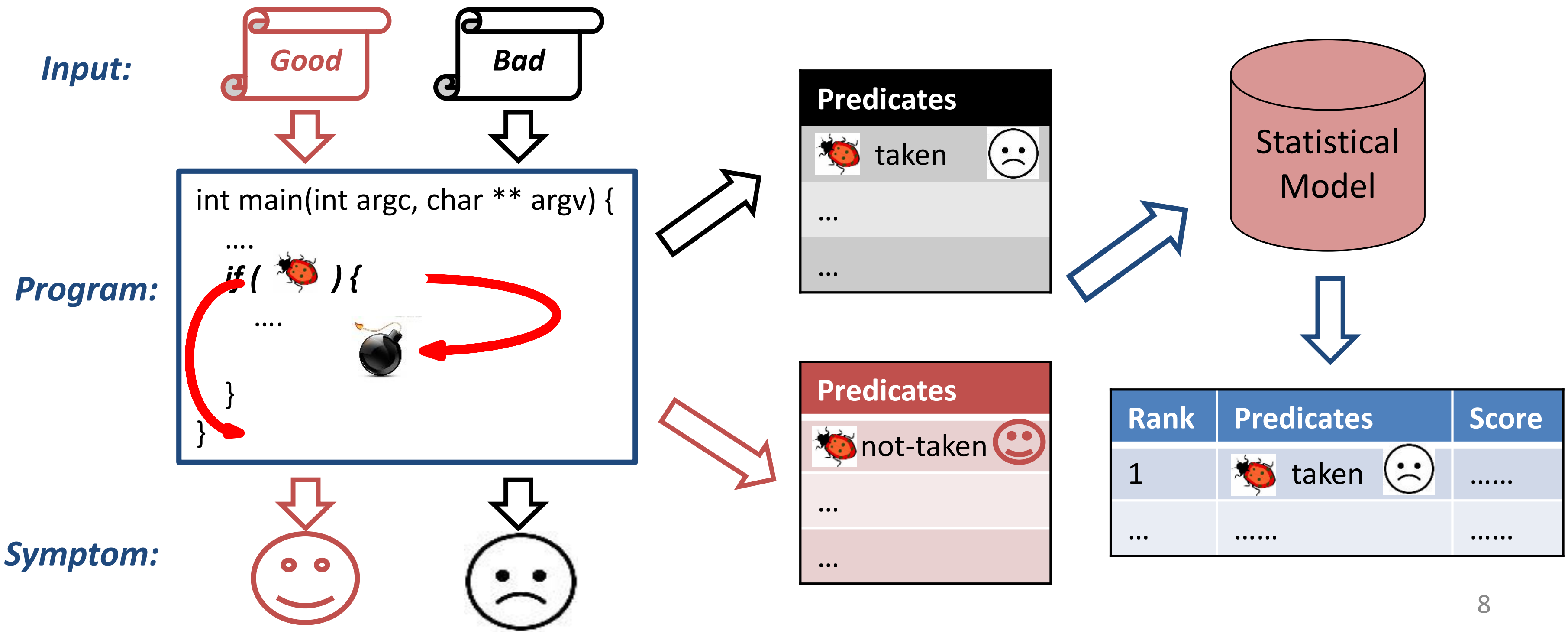
MySQL Bug 26527

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



```
int main(int argc, char ** argv) {
   ....
   if ( 🐞 ) {
      ....
   }
}
```

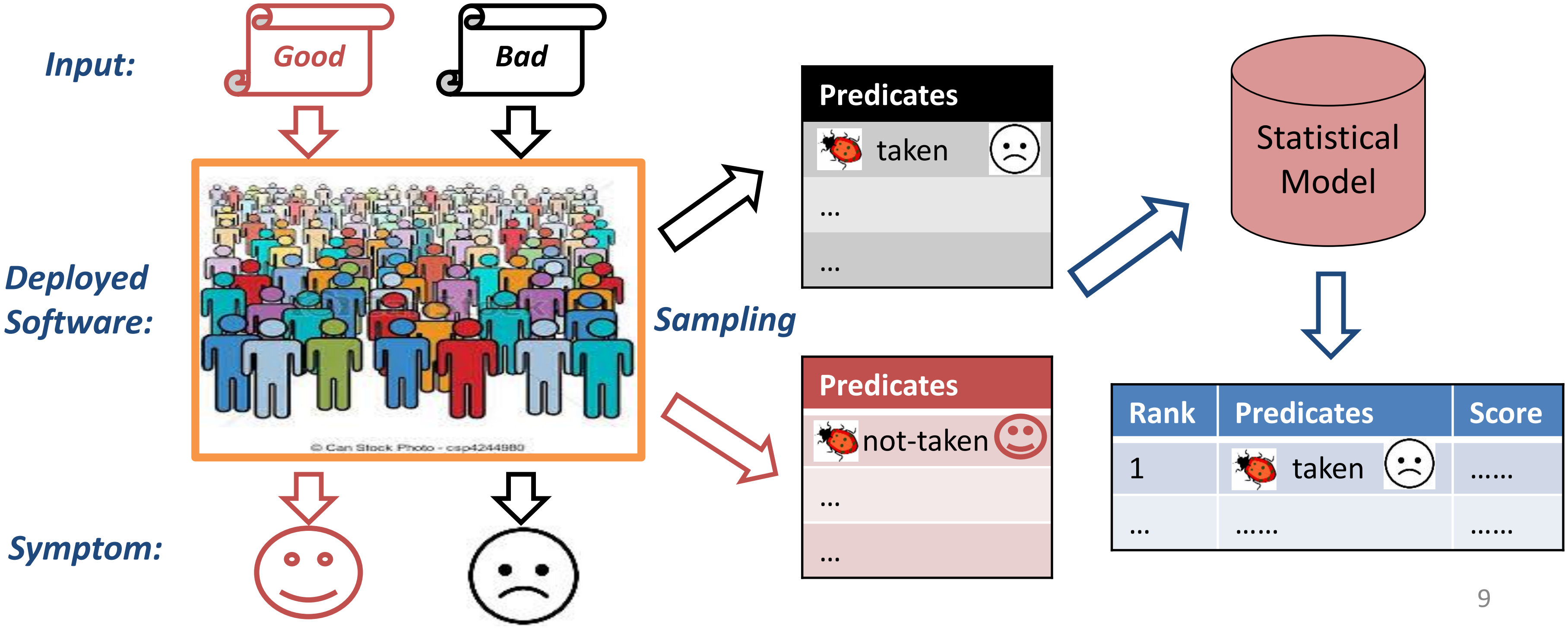
Predicates	
🐞 taken	☹
…	
…	

Predicates	
🐞 not-taken	☺
…	
…	

Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

How to Diagnose Functional Bugs

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



© Can Stock Photo - csp4244980

Predicates
🐞 taken ☹
…
…

Predicates
🐞 not-taken ☺
…
…

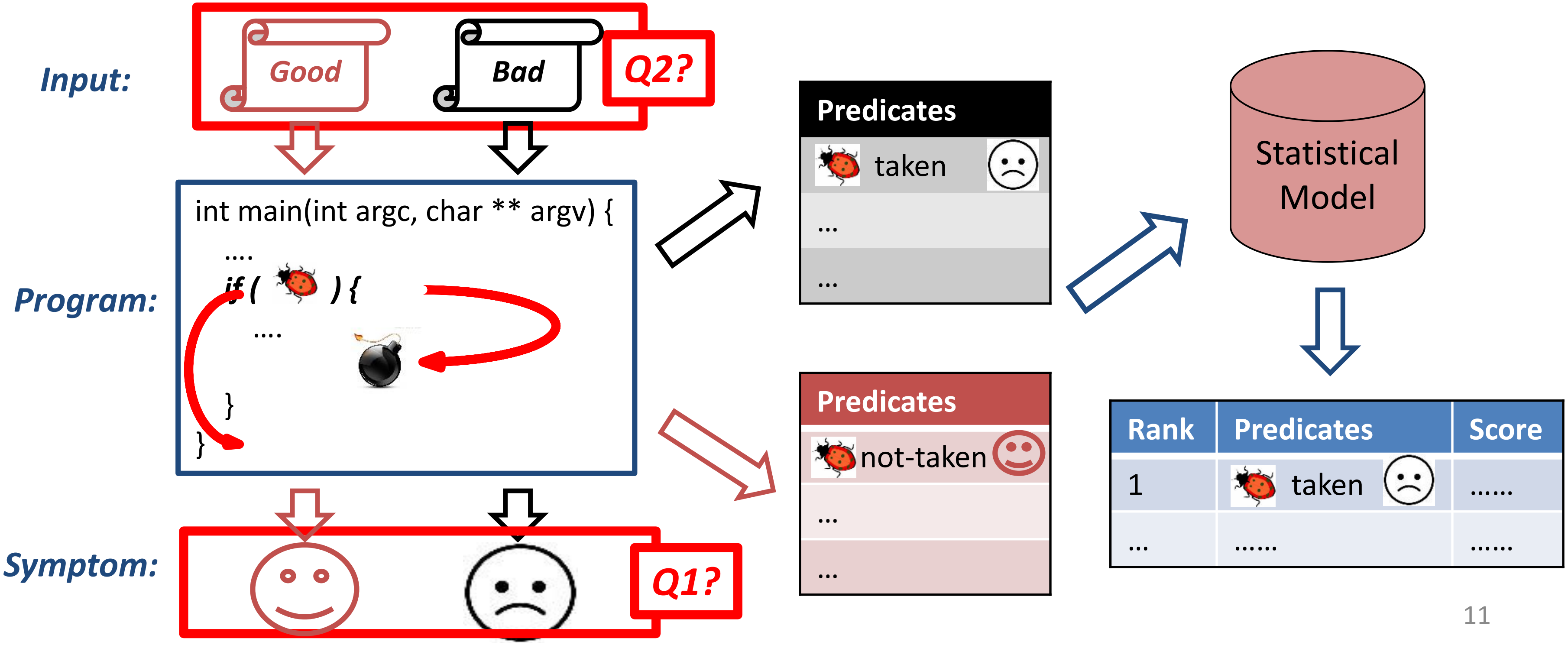
Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

Apply SD to Performance Diagnosis?

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

Is it Feasible?

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



```
int main(int argc, char ** argv) {
  ....
  if (    ) {
     ....
  }
}
```

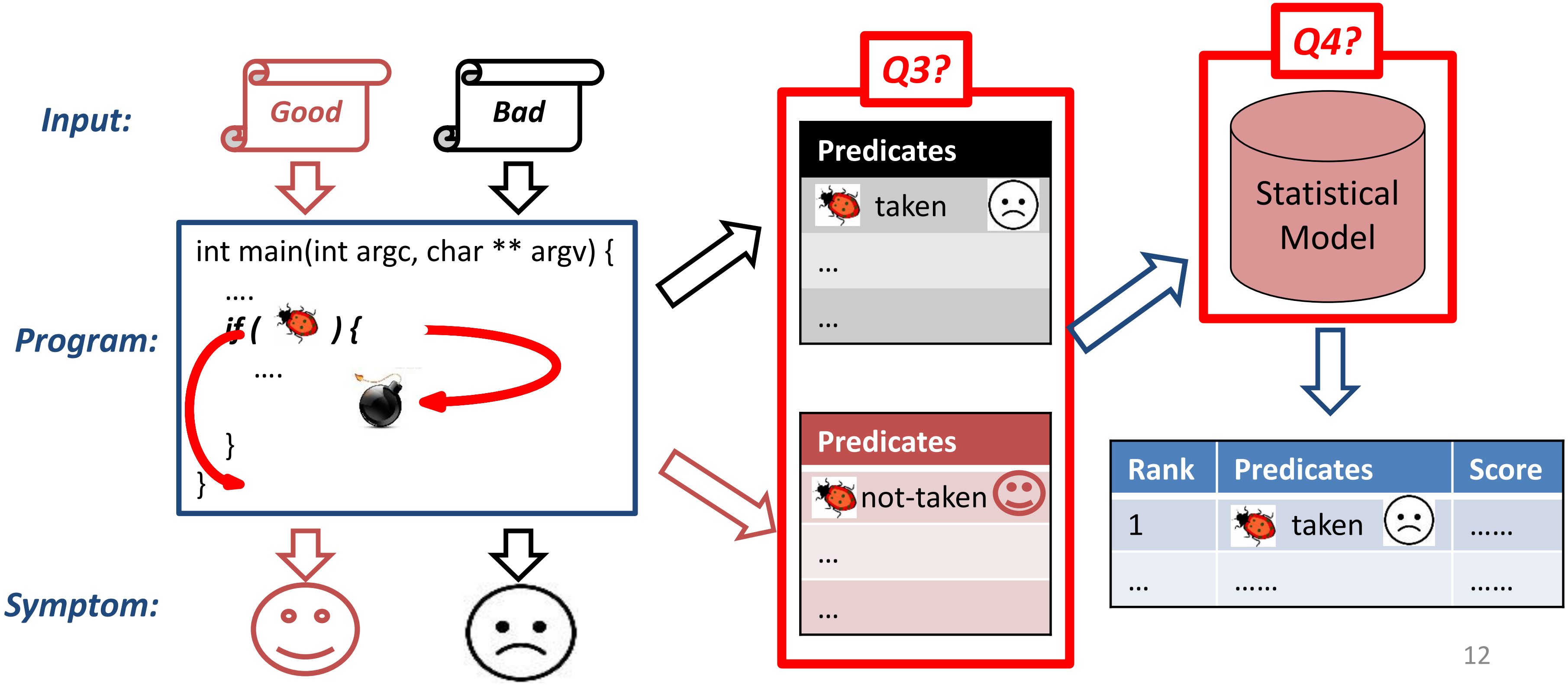
Predicates
taken
…
…

Predicates
not-taken
…
…

Rank	Predicates	Score
1	taken	……
…	……	……

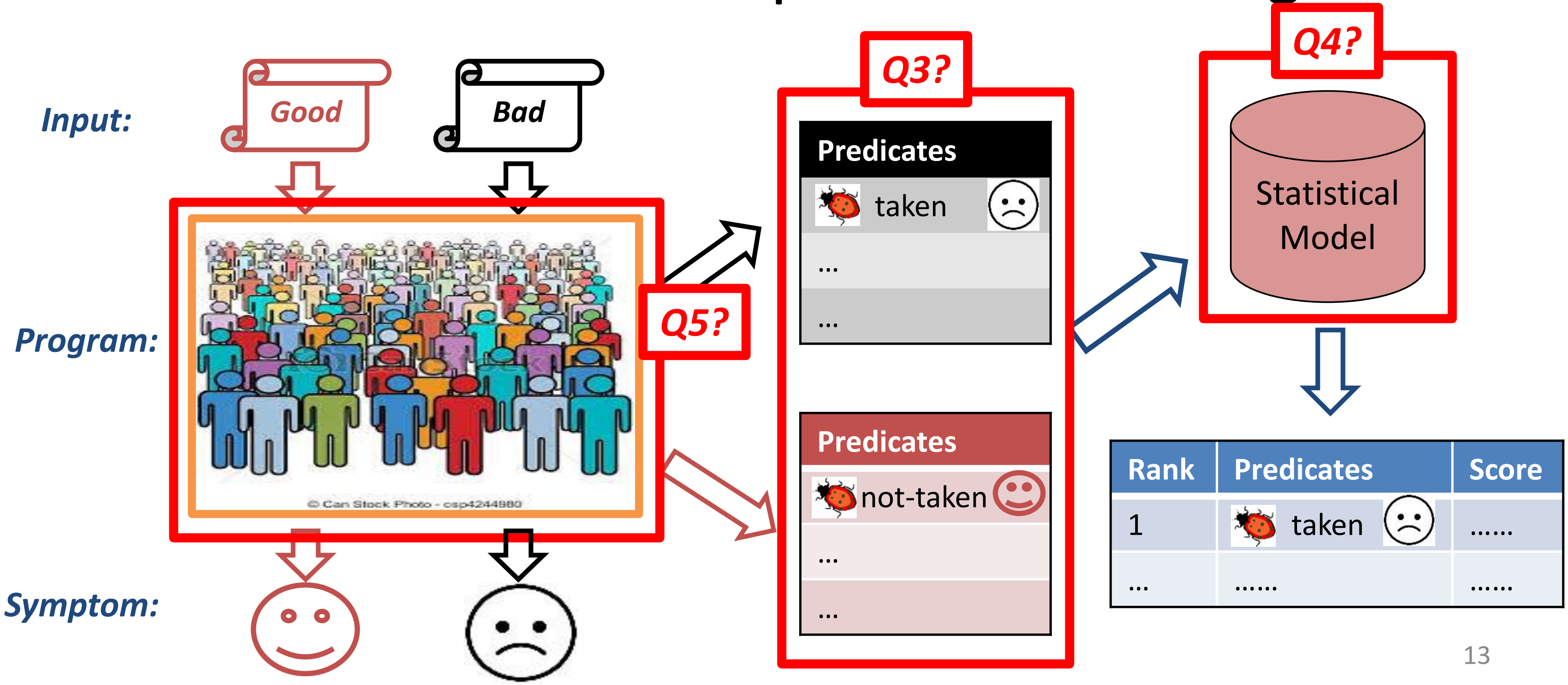
How to Apply?

- 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?



Input:

Good

Bad

Program:

Symptom:

Q5?

Q3?

Predicates

taken ☹

…

…

Predicates

not-taken ☺

…

…

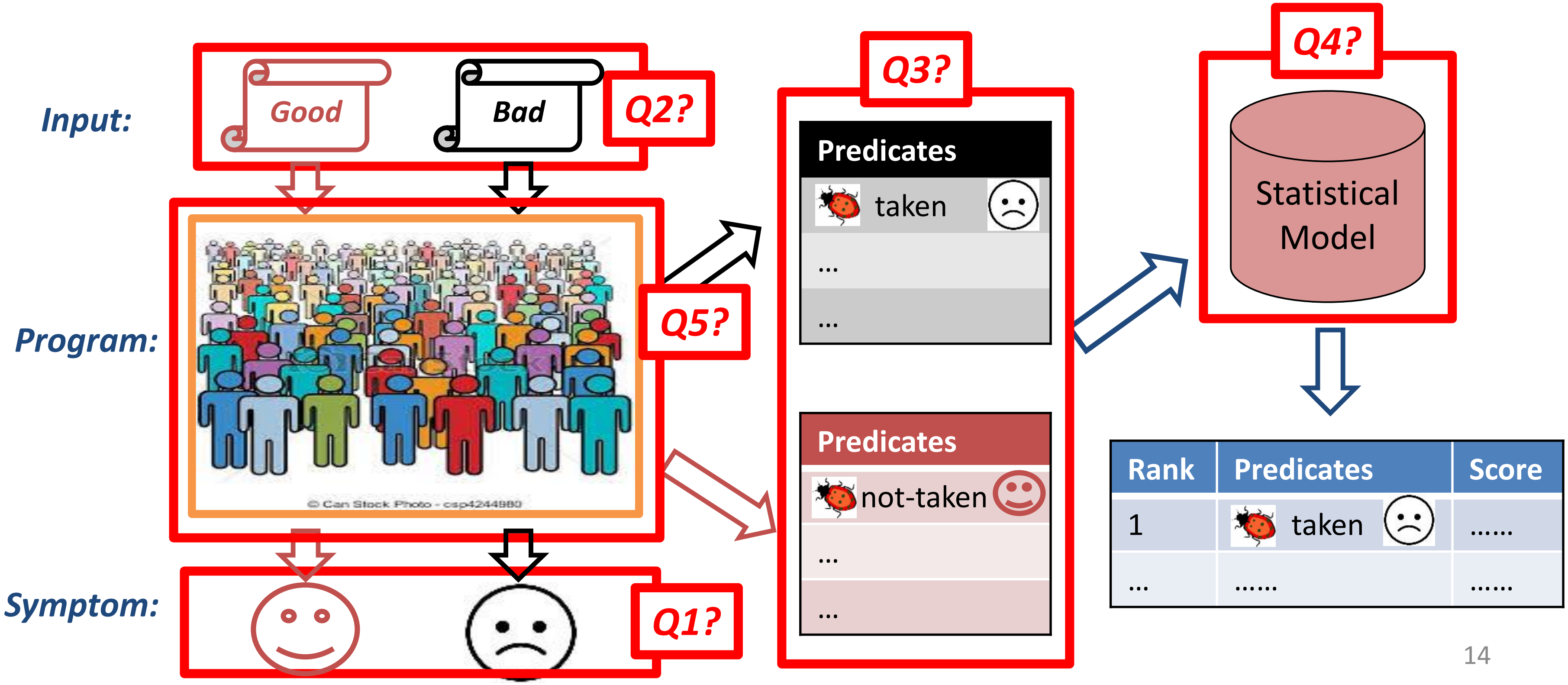
Q4?

Statistical Model

Rank	Predicates	Score
1	taken ☹	……
…	……	……

Contributions (I)

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



Input:

Good

Bad

Q2?

Program:

Q5?

Symptom:

☺ ☹

Q1?

Q3?

Predicates	
🐞 taken	☹
…	
…	

Predicates	
🐞 not-taken	☺
…	
…	

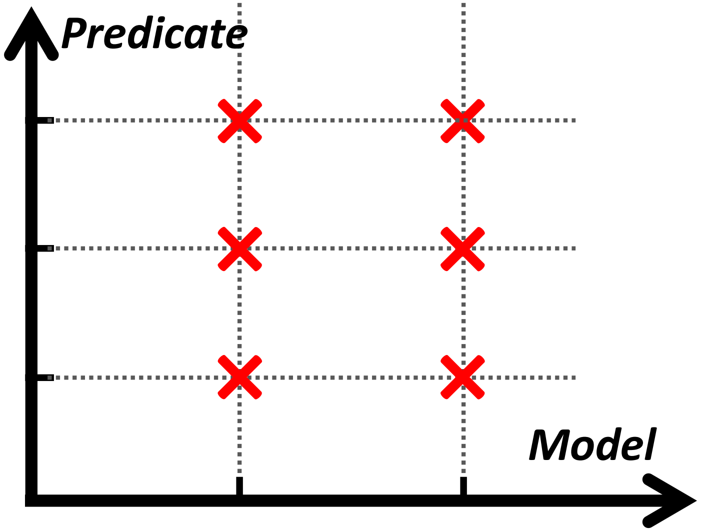
Q4?

Statistical Model

Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

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

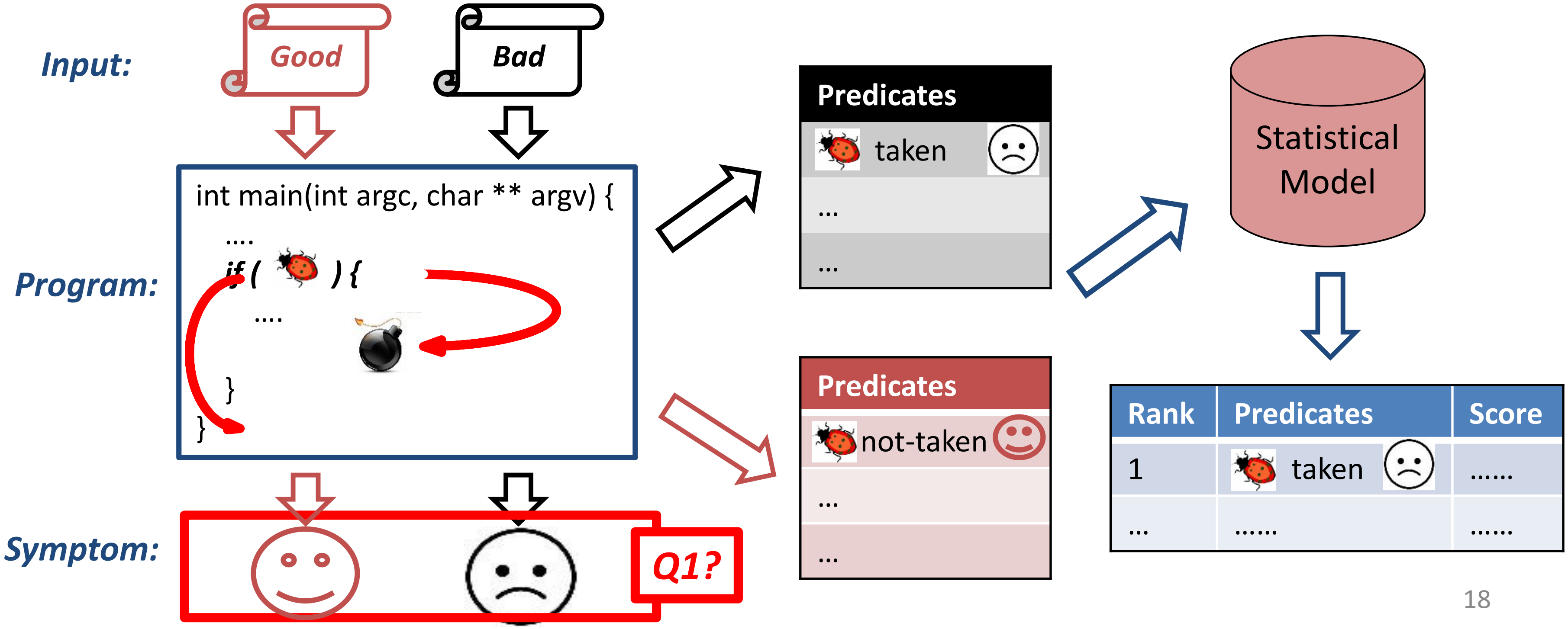
Total: 110

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?



Input: *Good* *Bad*

Program:

```
int main(int argc, char ** argv) {
   ....
   if ( 🐞 ) {
      ....
   }
}
```

Symptom: ☺ ☹ *Q1?*

Predicates
🐞 taken ☹
…
…

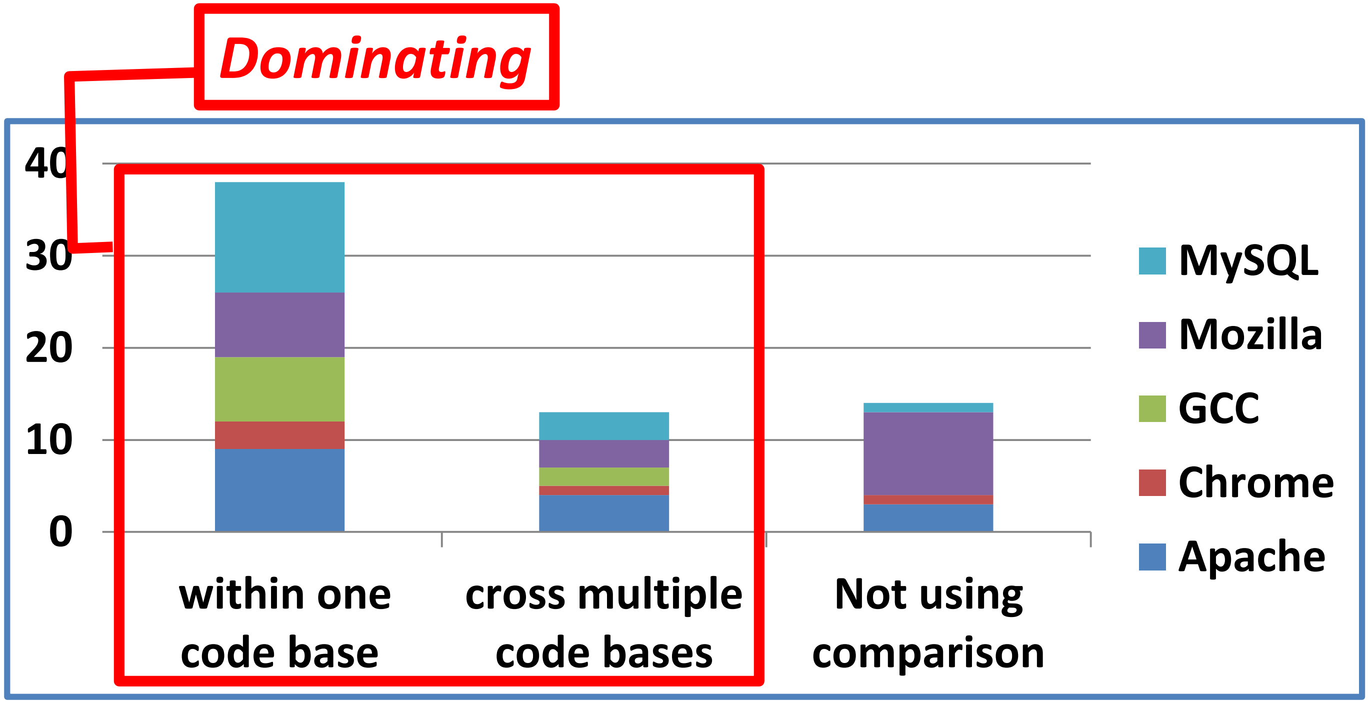
Predicates
🐞 not-taken ☺
…
…

Statistical Model

Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

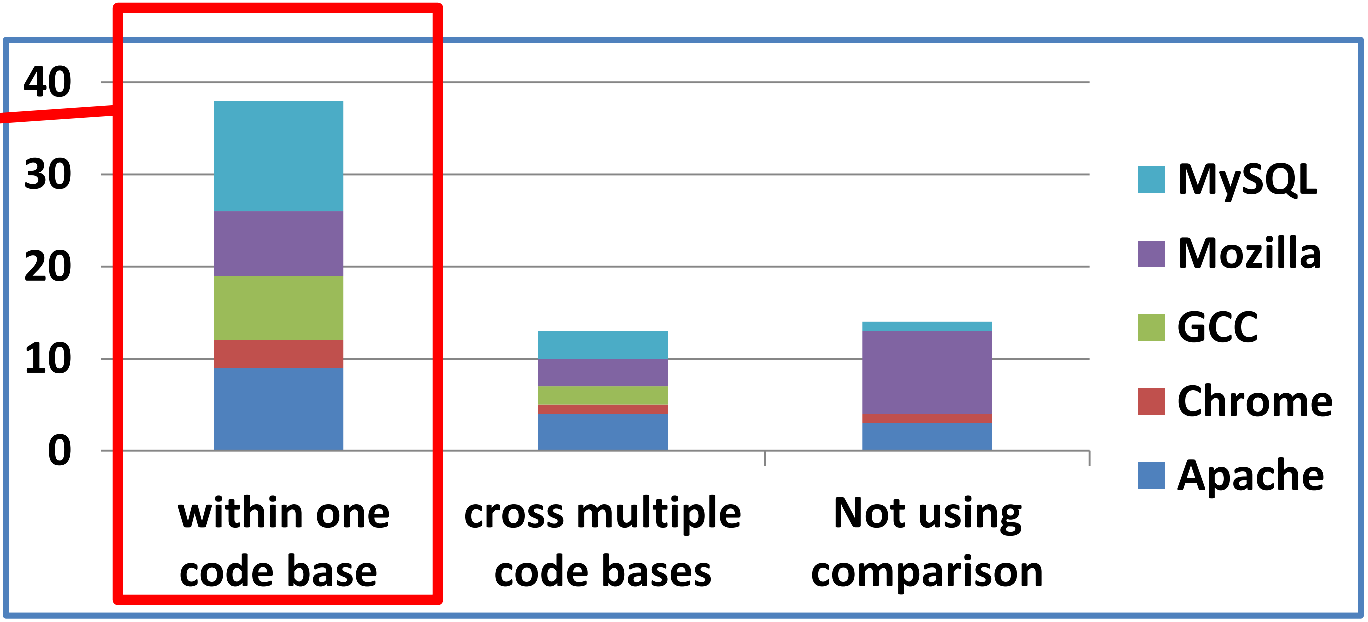
Q1: How to Identify Failure Runs?

- The majority is observed through comparison



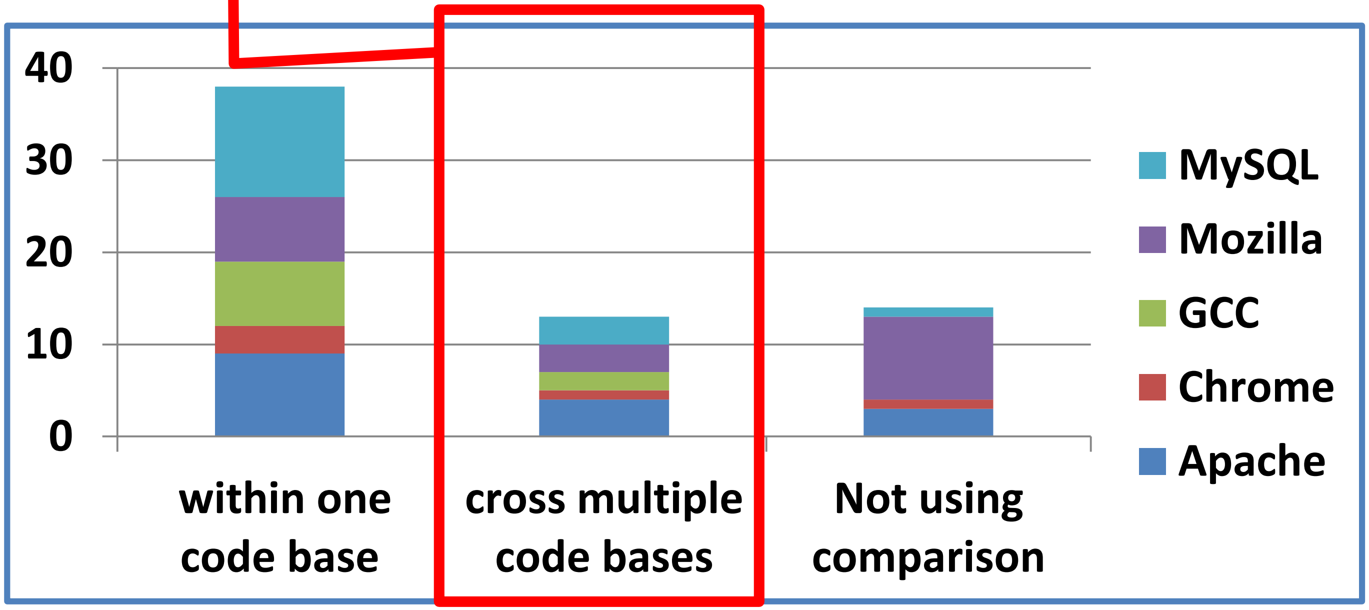
Comparison within One Code Base

- 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

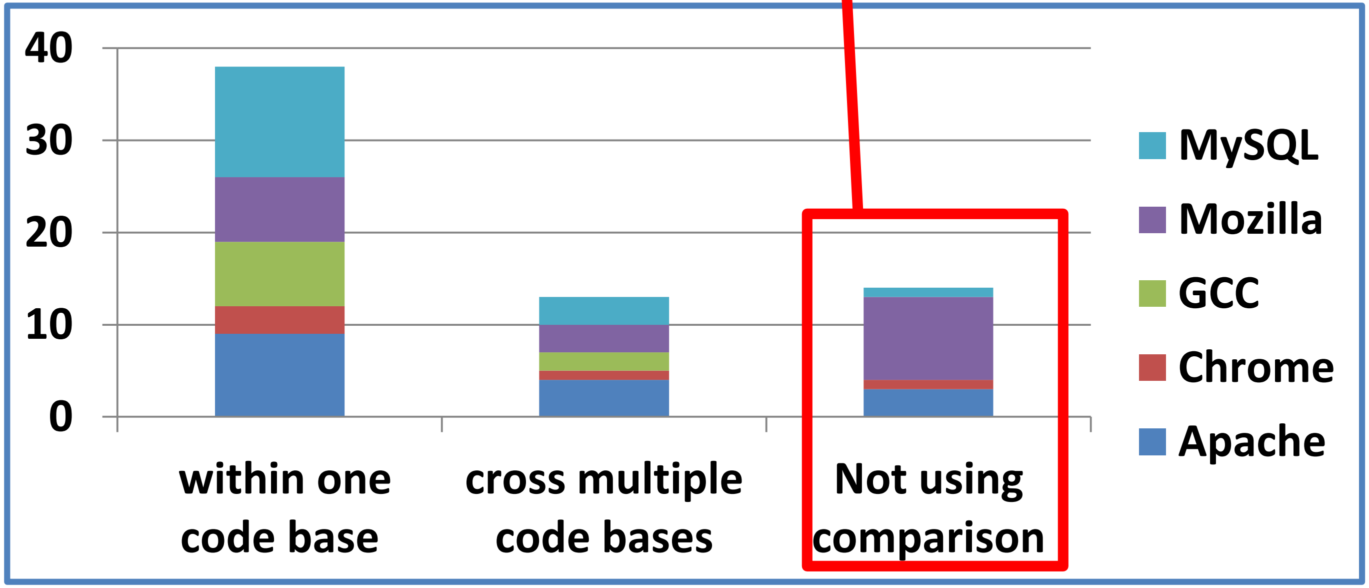
Mozilla#299742: “it frozen the GUI to crawl”

Mozi vith the page”

MySQL46461: “causing the test suit to fail due to timeout”

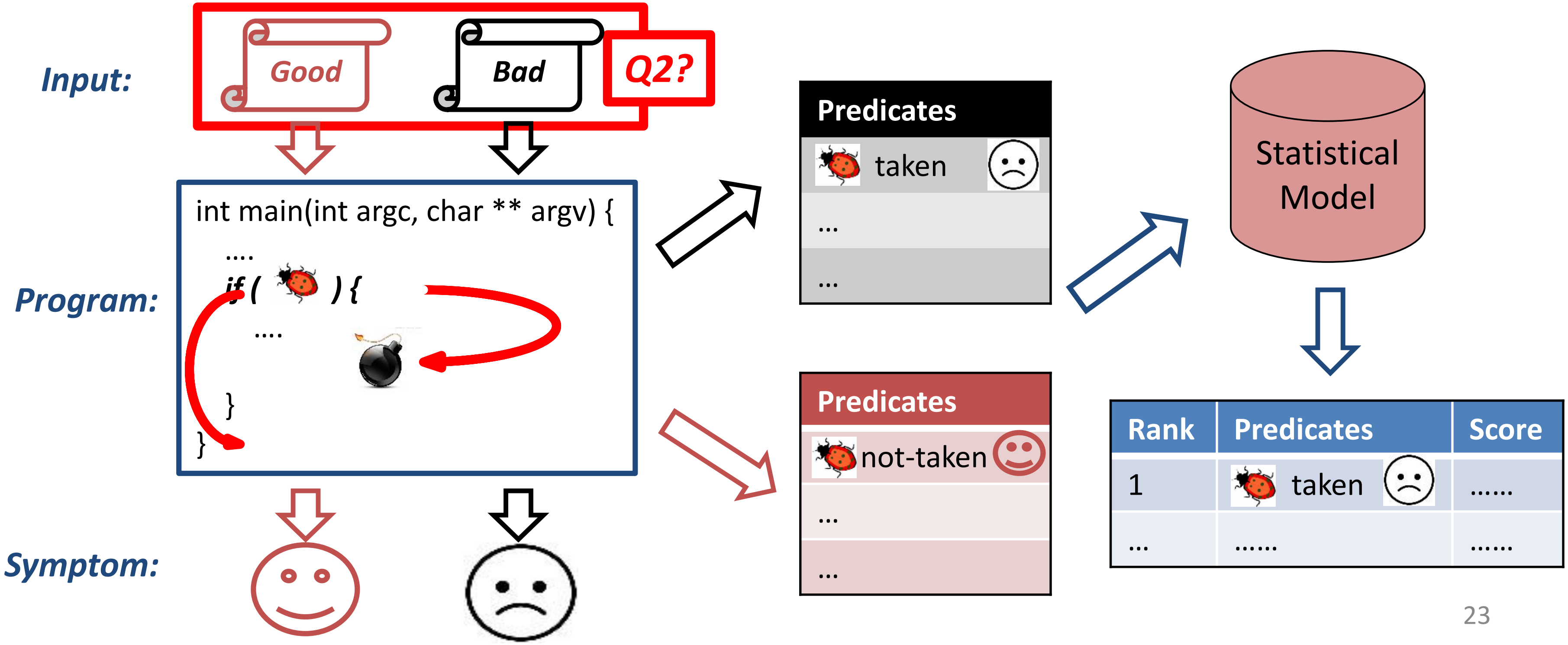
Easy to tell failures from successes!

Non-tolerable



Is It Feasible? (Part II)

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



Input:

Good *Bad* *Q2?*

Program:

```
int main(int argc, char ** argv) {
   ....
   if ( 🐞 ) {
      ....
                 💣
   }
}
```

Symptom:

☺ ☹

Predicates
🐞 taken ☹
…
…

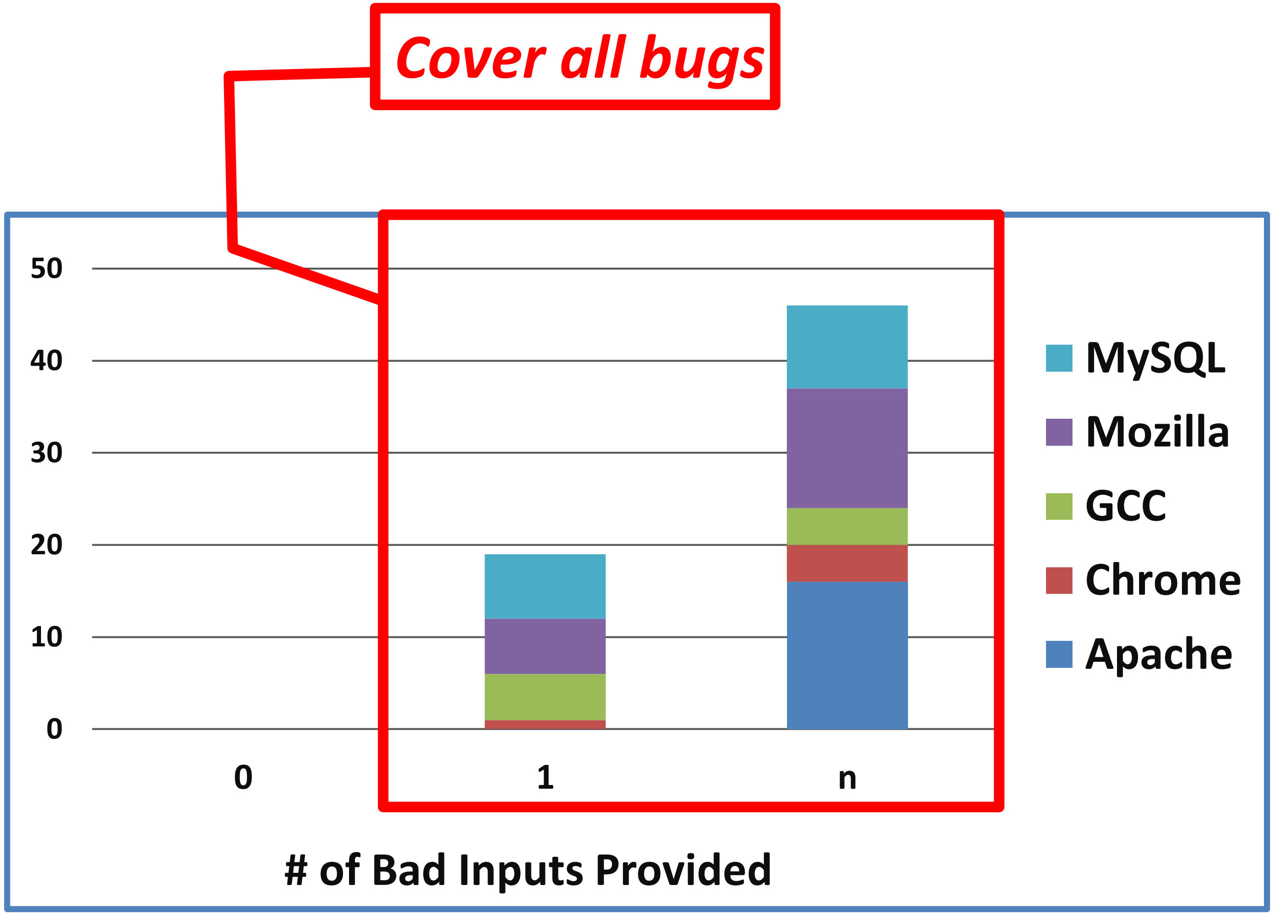
Predicates
🐞 not-taken ☺
…
…

Statistical Model

Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

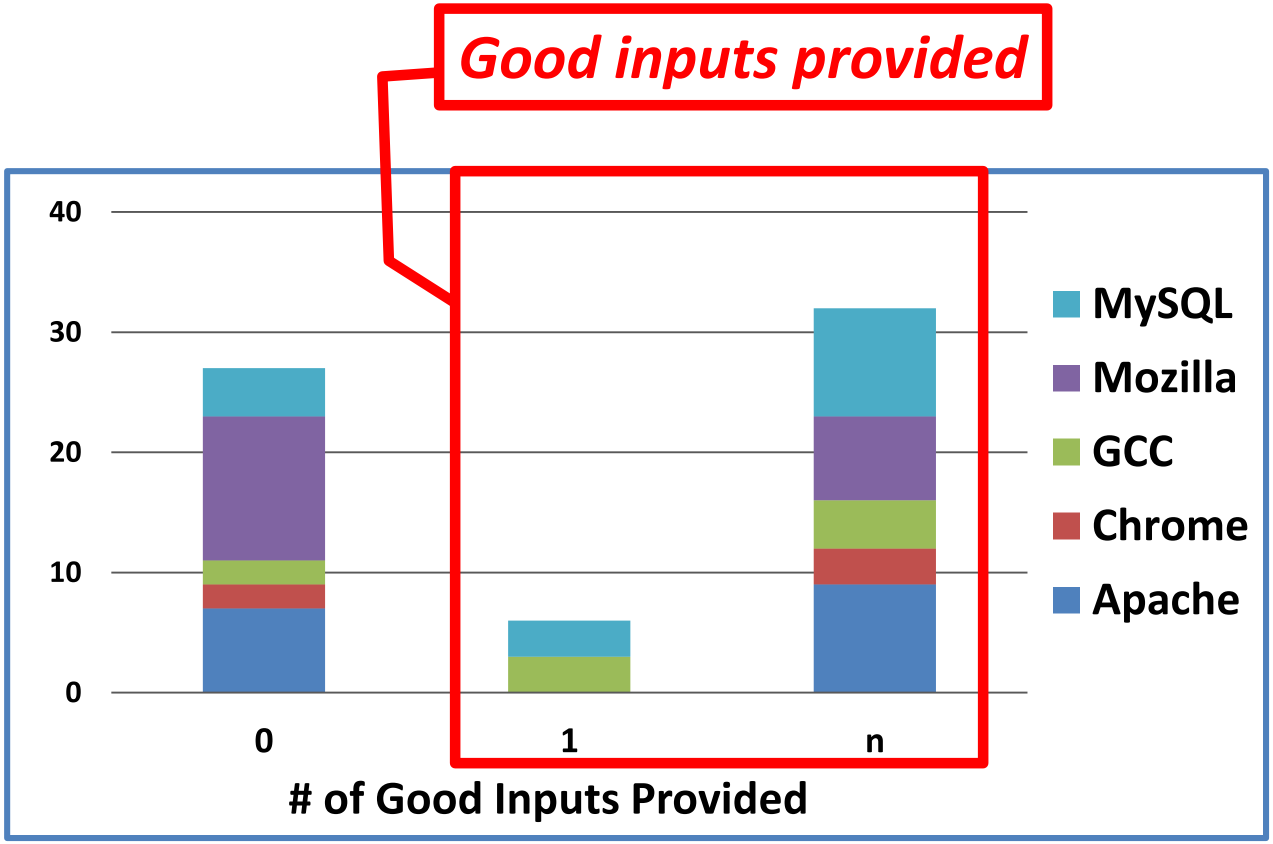
How to Obtain Bad Inputs?

- Bad inputs are provided in all bug reports



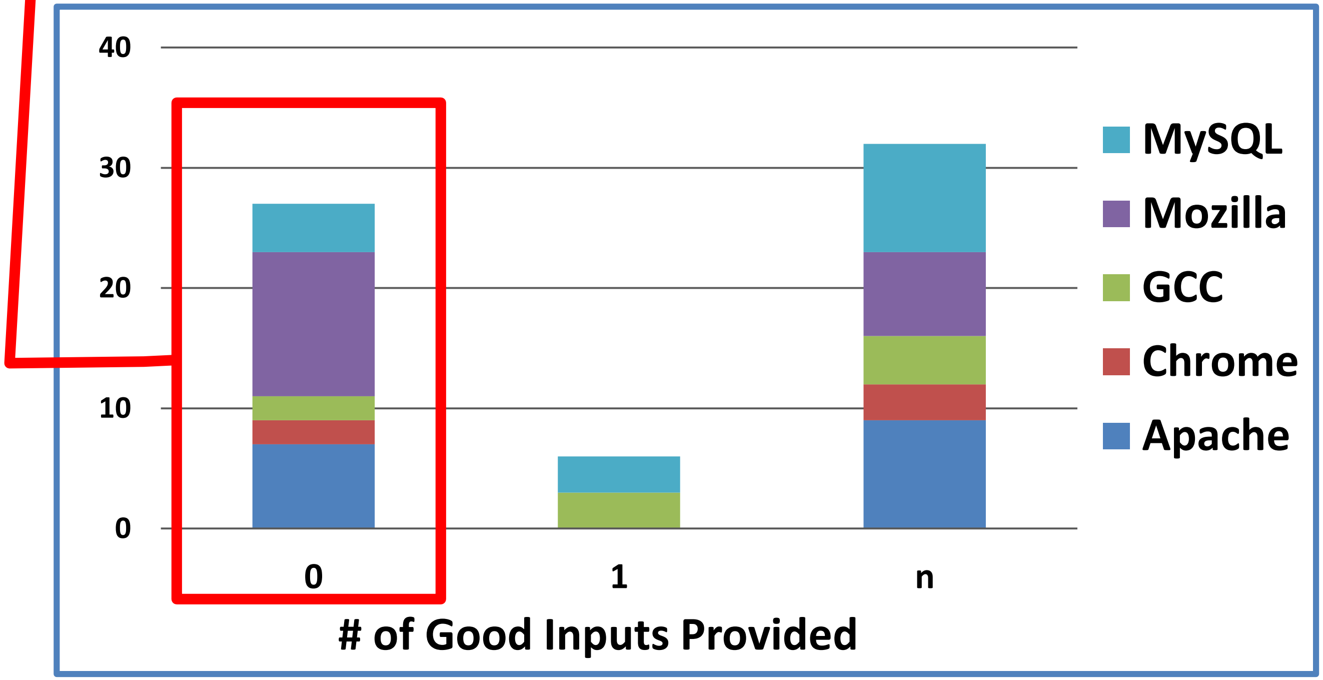
How to Obtain Good Inputs? (I)

- 60% contain good inputs



How to Obtain Good Inputs? (II)

Easy to design



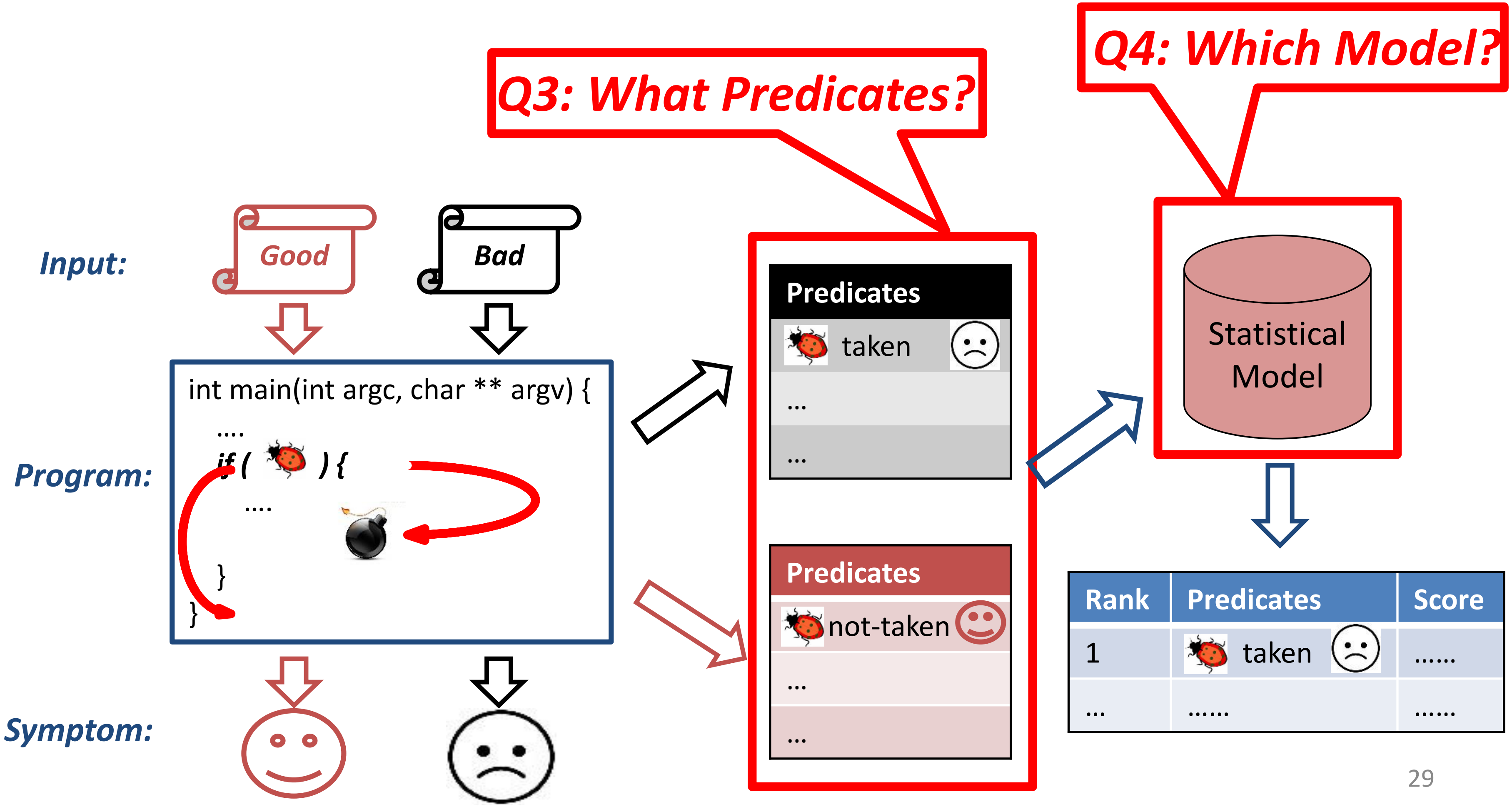
Other Findings and Implications

- 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



Q3: What Predicates?

Q4: Which Model?

Input:

Good

Bad

Program:

```
int main(int argc, char ** argv) {
    ….
    if ( 🐞 ) {
        ….
              💣
    }
}
```

Symptom: ☺ ☹

Predicates
🐞 taken ☹
…
…

Predicates
🐞 not-taken ☺
…
…

Statistical Model

Rank	Predicates	Score
1	🐞 taken ☹	……
…	……	……

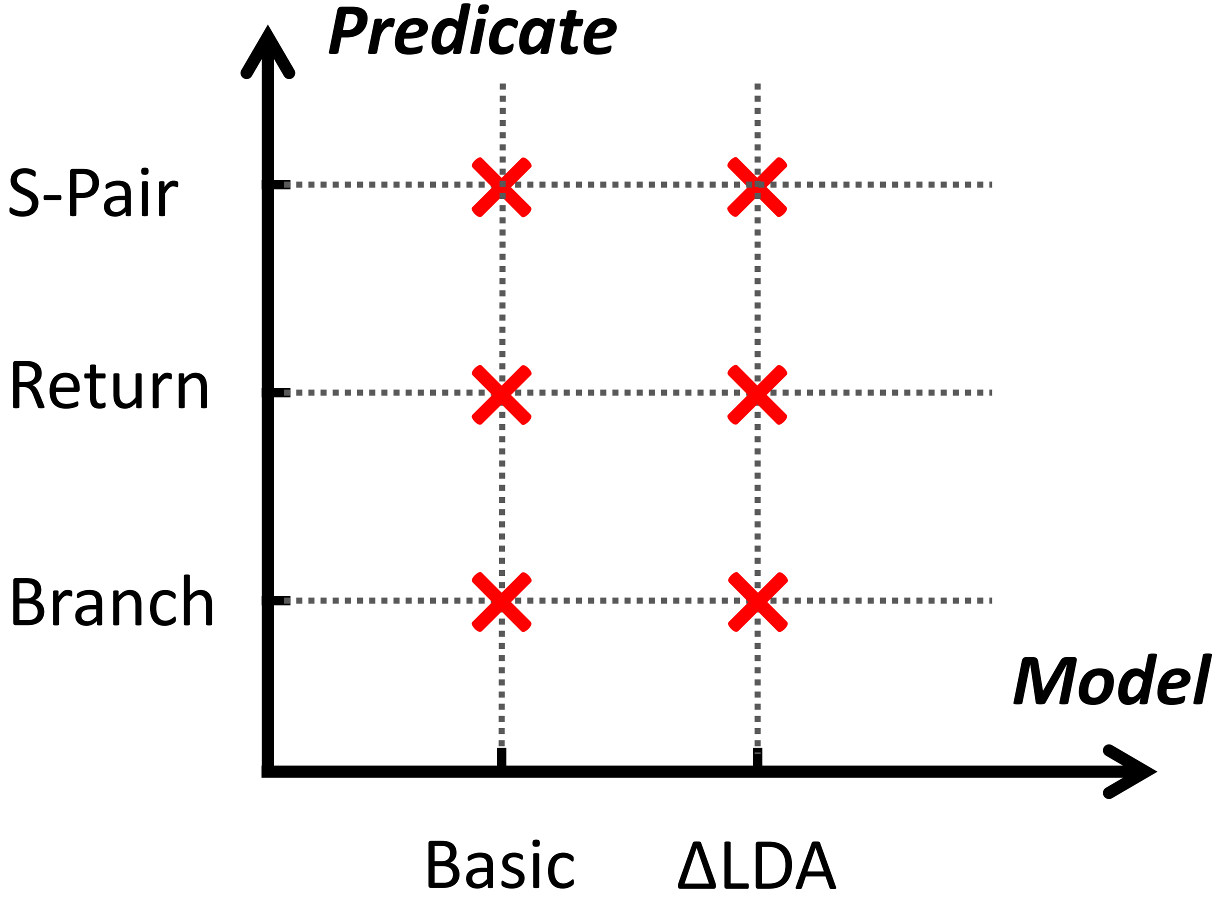
Design

- Study widely used predicates and models

```
int x, y;
…
x = …;
```

```
n=fprintf(…);
```

```
if (p)  …
else  ….
```



	Basic	ΔLDA
S-Pair	✗	✗
Return	✗	✗
Branch	✗	✗

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

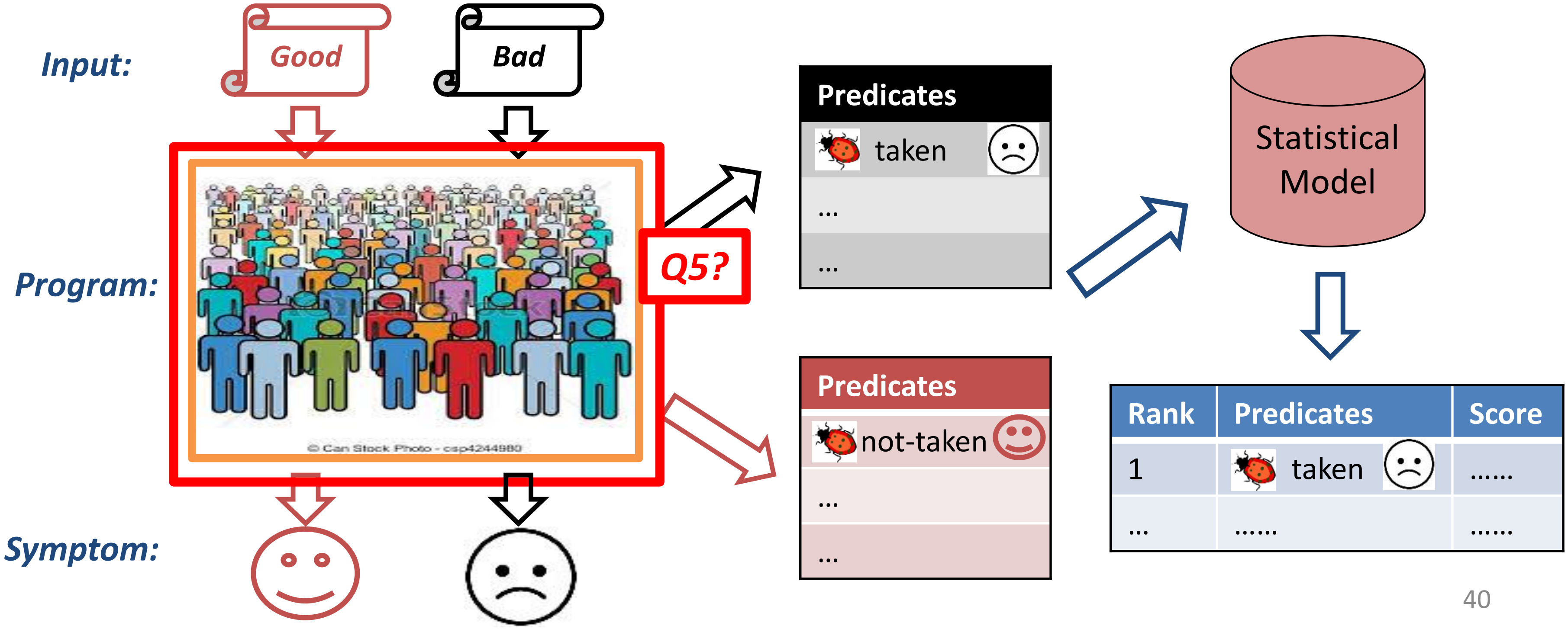
	Basic Model			ΔLDA			Profiler
BugID	Branch	Return	S-pair	**Branch**	Return	S-pair	
Mozilla258793	√1	-	/	-	-	/	-
Mozilla299742	√1	-	/	-	-	/	-
Mozilla347306	-	-	-	√1	√1	√1	√1
Mozilla416628	-	-	-	√1	-	√1	√1
MySQL15811	-	-	/	√1	√1	/	√1
MySQL26527	√1	-	/	-	-	/	-
MySQL27287	-	-	/	√1	-	/	√1
MySQL40337	√1	-	/	-	-	/	-
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.



Predicates
taken ☹
…
…

Predicates
not-taken ☺
…
…

Rank	Predicates	Score
1	taken ☹	……
…	……	……

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
[illegible]	√1	2.39%	[illegible]
[illegible]	√1	4.27%	500
[illegible]	√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	√1	2.39%	
Mozilla299742	√1	4.27%	
Mozilla347306	√1	1.42%	
Mozilla416628	√	[illegible]	
MySQL15811	√1	2.25%	
MySQL26527	√1	6.05%	
MySQL27287	√1	3.02%	
MySQL40337	√1	2.69%	
MySQL42649	√2	6.10%	
MySQL44723	√1	3.16%	
Apache3278	-	0.23%	
Apache34464	√3	0.18%	
…	…	…	…

10 X 10000 ??

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
…	…	…	…

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

Thanks a lot!