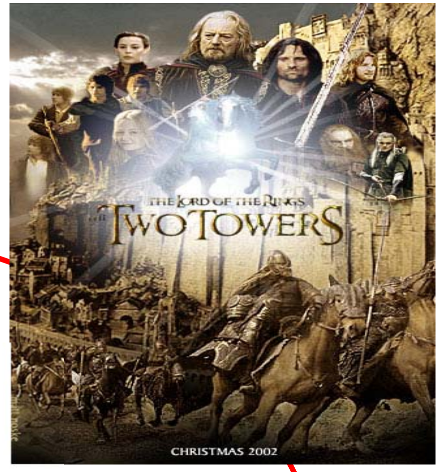
뮤즈 (MUSE): 프로그램의 수많은 돌연변이들을 활용한 버그 위치 추정 기법

문석현, 김윤호, <u>김문주</u>
Software Testing & Verification Group (SWTV)
CS Dept., KAIST

유신
Centre of Research on Evolution, Search and Testing (CREST)
CS Dept. Univ. College London (UCL), UK

반지원정대

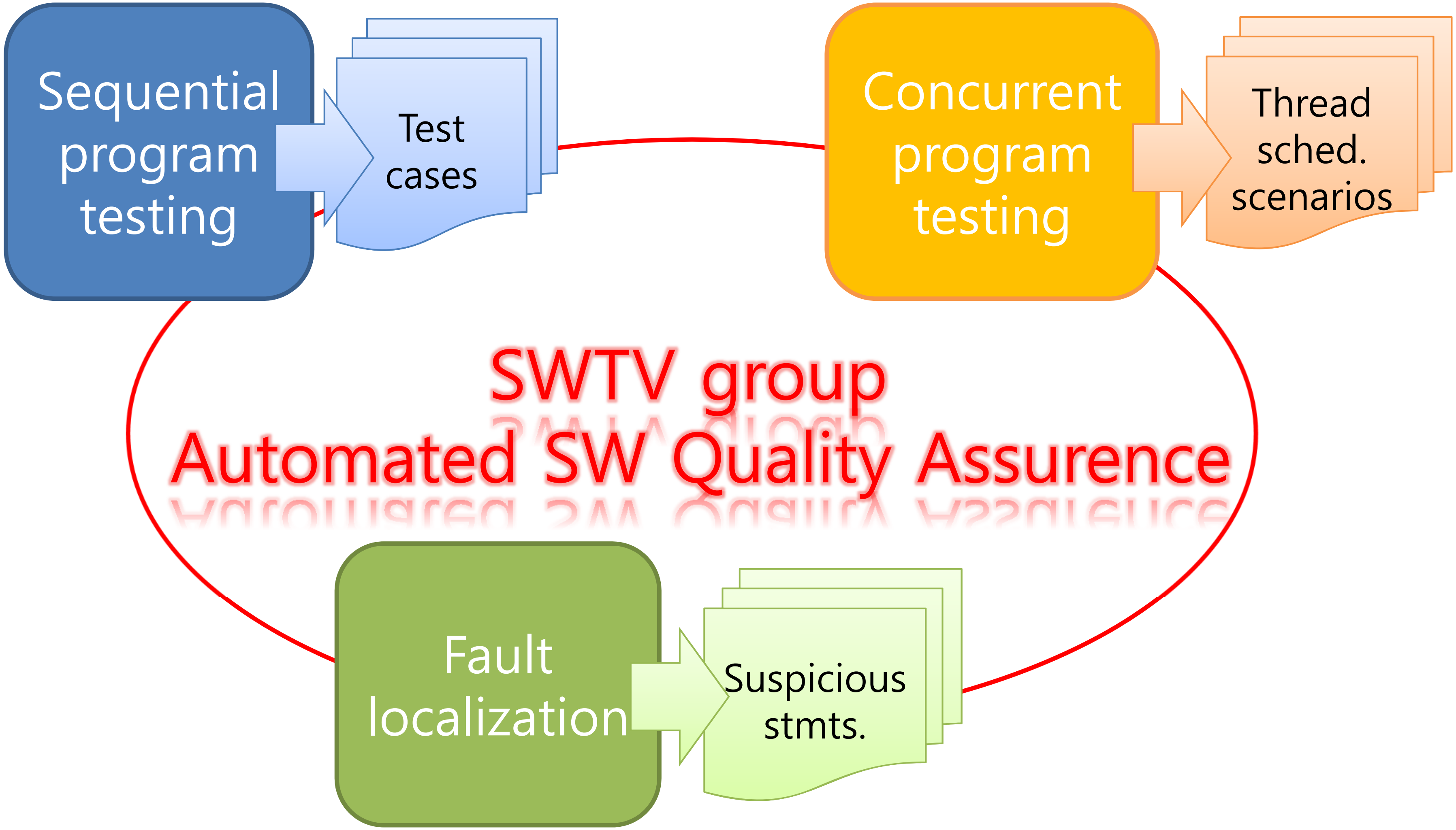
Peter Jackson
반지의 제왕 3부작



두개의 탑

왕의 귀환

Research Directions @ SWTV



Motivation

- Developers have spent a large amount of time in debugging.
- One of the most **laborious task** of debugging activity is to locate the cause of failures (i.e., fault), which is called ***fault localization***.

Test case #2 (0,-4)

```
int max = 0;
void Setmax(int x, int y) {
  max -= x; // should be 'max=x;'
  if (max<y) {
    max = y;
    if(x*y<0)
      print ("diff. sign"); }
  print ("%d", max);
  Assert (x>=y? max == x : max == y); }
```

Test result: **fail**

Inspecting line by line

== Laborious task

- Research Goal**:** To develop **automated fault localization techniques** that assist developers effectively **locate the cause of program failures** (i.e., fault).

Contributions

- We have developed techniques that automatically prioritize likely faulty statements using dynamic information of test executions.
  - **MU**tation-ba**SE**d fault localization technique (MUSE) that utilizes mutation analysis to localize faults.
    - **A novel approach using mutation analysis for the fault localization.**
    - **Highly precise.**
      - **MUSE is 5.6 times more precise than the state-of-art fault localization technique (ranks the faulty statement among the top 1.65% of executed statements).**
    - **Widely applicable.**
      - **MUSE only requires source code of target program and test suite.**

Related Work

- Program slicing [Weiser, ICSE1981]
  - analyzing program dependencies.
- Delta debugging [Zeller, ESEC/FSE2002]
  - analyzing differences between states of a failing execution and those of a passing execution.
- Spectrum-Based Fault Localization (SBFL) [Jones et al., ICSE2002]
  - Spectrum: a set of program entities (e.g., statements) executed by a test case.
  - computing suspiciousness of each entity based on program spectra.
  - E.g., $Susp_{Jaccard}(s) = \frac{|f_P(s)|}{|f_P(s)|+|p_P(s)|}$ where $f_P(s)$ and $p_P(s)$ are a set of failing and a set of passing test cases that execute s in a target program P, respectively.

`int max = 0;` `void Setmax(int x, int y) {`	Spectrum of test cases					*Jaccard*	
	tc 1 (3,1)	tc 2 (5,-4)	tc 3 (0,-4)	tc 4 (0,7)	tc 5 (-1,3)	Susp.	Rank
`1: max -= x; // should be 'max=x;'`	●	●	●	●	●	0.40	5
`2: if (max<y) {`	●	●	●	●	●	0.40	5
`3:  max = y;`	●	●		●	●	0.50	2
`4:  if(x*y<0)`	●	●		●	●	0.50	2
`5:    print ("diff. sign"); }`		●			●	0.33	6
`6: print ("%d", max); }`	●	●	●	●	●	0.40	5
Pass / Fail status	Fail	Fail	Pass	Pass	Pass		

- Developers can find the faulty statement by examining 83.3% (=5/6) of executed statements.

Spectrum-Based Fault Localization

- SBFL outperforms other kinds of fault localization techniques (i.e., program slicing, delta debugging) [Jones et al., ASE2005],
  - Program slicing, delta debugging, etc.
  - Thus, many researchers have focused on improving the precision of SBFL.

- However, SBFL has also been criticized for its impractical accuracy [Parnin et al. ISSTA 2011].
  - The rank of the faulty statement is too low to use SBFL practically.
  - Comparison results of SBFL techniques on 7 programs from SIEMENS benchmark.

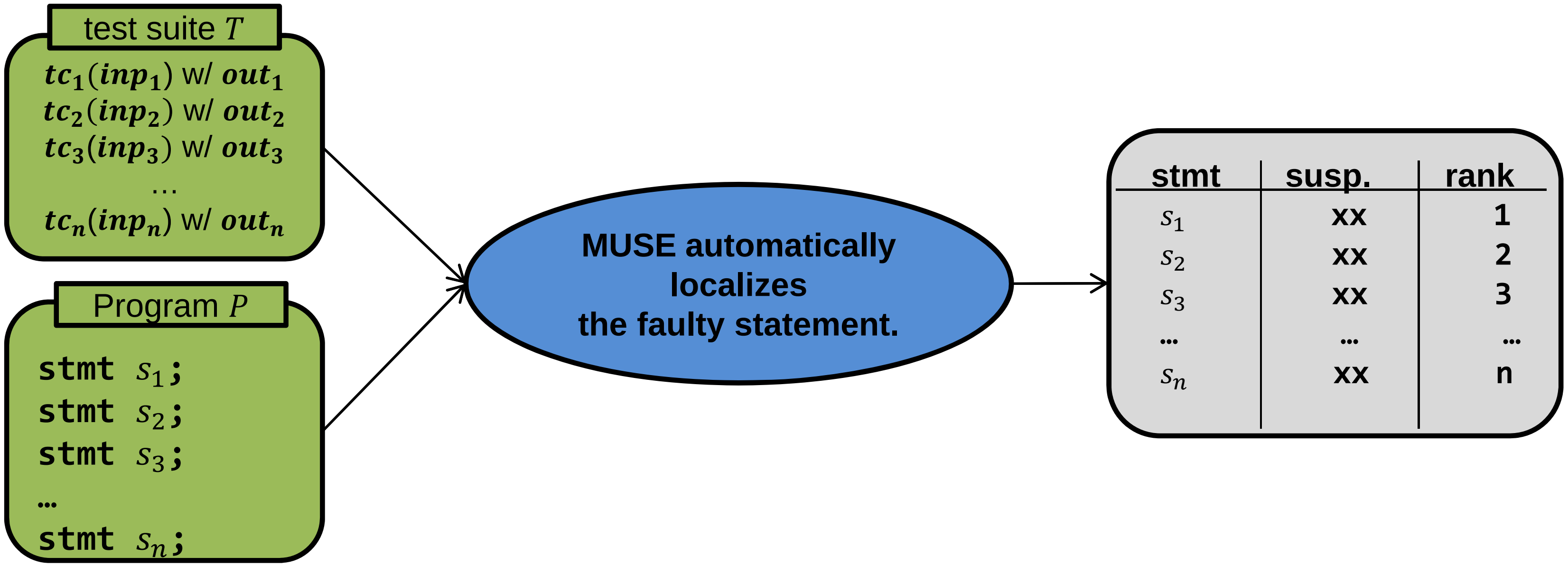
SBFL Technique	% of executed stmts examined	SBFL Technique	% of executed stmts examined
Op2	**15.75**	Cohen	21.20
Op1	15.79	CBI Log	21.90
M2	16.91	CBI Sqrt	22.00
Ochiai	18.42	Ochiai2	24.01
Amean	19.61	Binary	27.91
Hmean	19.72	Russell	27.87
Ample2	20.25	Overlap	27.96
Jaccard	20.72	Ample	26.95
Rogot2	21.45	Scott	36.98
Tarantula	21.59	**Fleiss**	**37.23**

*extracted from [Naish et al., TOSEM2011]

➔ **An innovative approach is required to improve the precision!**

Our Approach

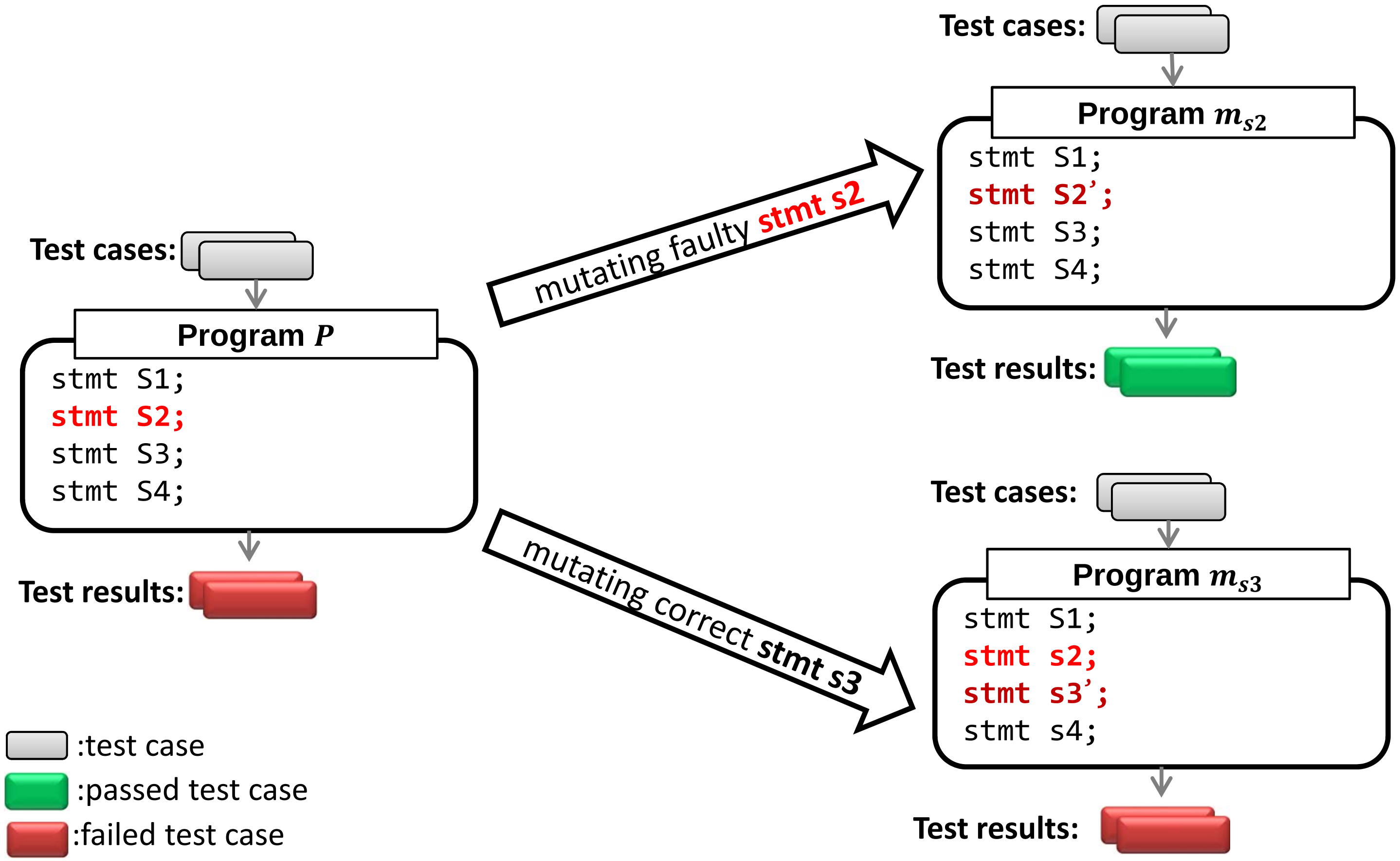
- I propose MUSE (**MU**tation-ba**SE**d fault localization technique), a new fault localization technique based on mutation analysis.



  – MUSE localizes faulty statements based on ***two key conjectures***.

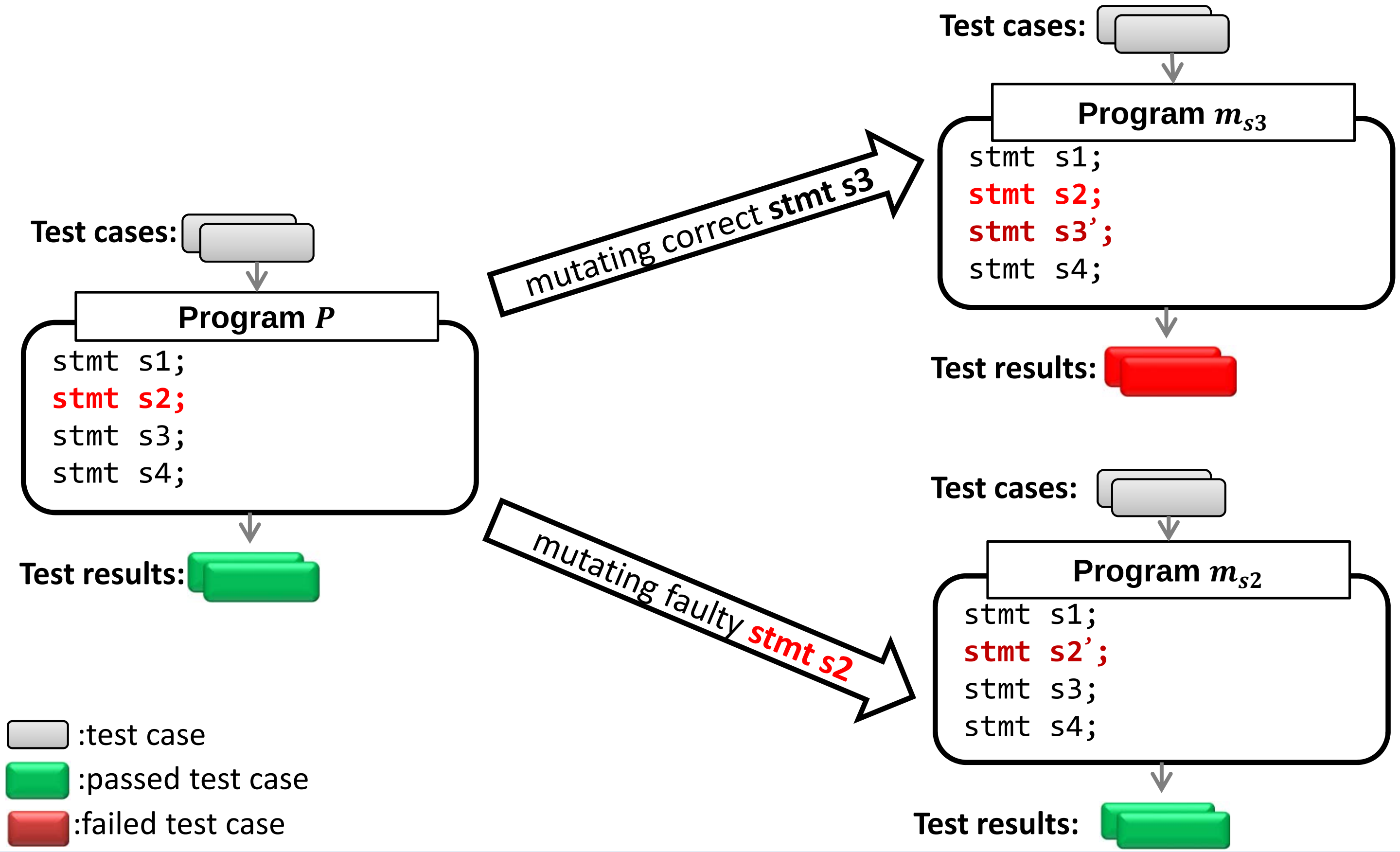
Key Conjecture Ⅰ

- **Conjecture Ⅰ** : mutating faulty statements is more likely to make failed tests pass than mutating correct statements.



Key Conjecture Ⅱ

- **Conjecture Ⅱ** : mutating correct statements is more likely to make passed tests fail than mutating faulty statements.



MUSE: Suspiciousness Metric

- Based on the two conjectures, the suspiciousness metric of μ for a statement s in a program P is defined as: $Susp_{\mu}(s) = \alpha_s - \beta_s$
  - α_s : The average # of failing tests that become passing ones for all mutants on s.
  - β_s : The average # of passing tests that become failing ones for all mutants on s.

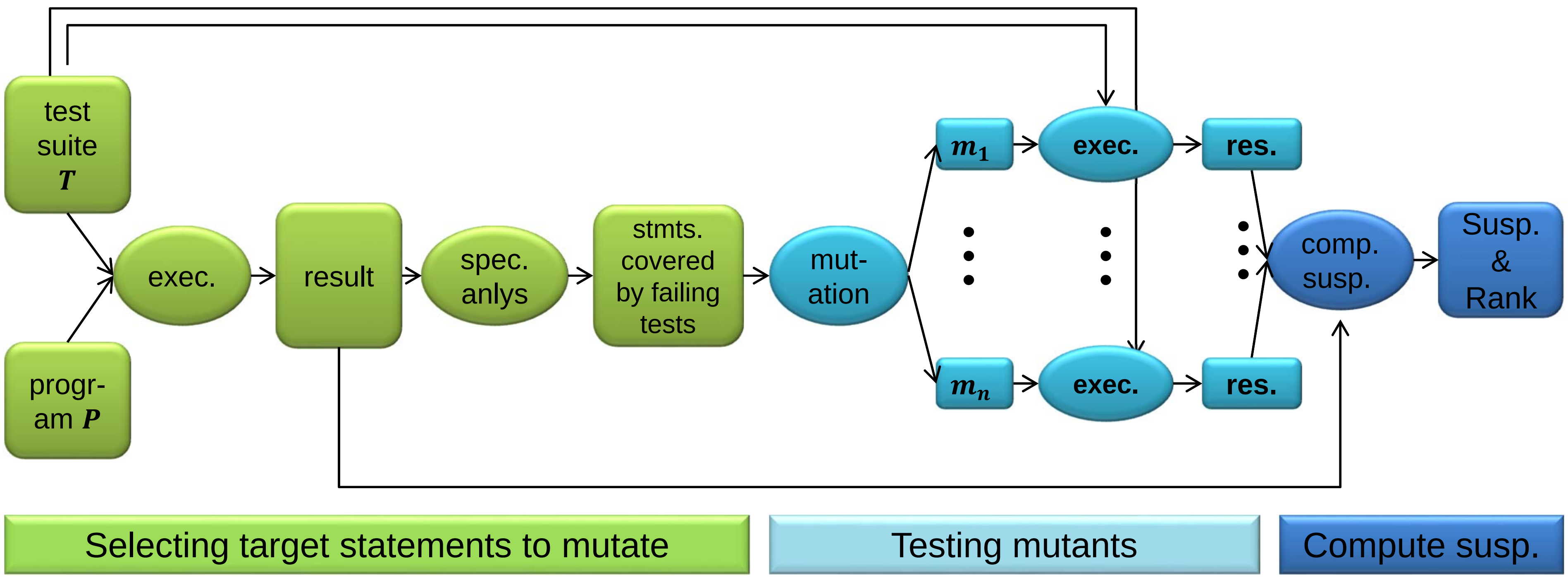
- Very detailed MUSE metric
  - $Susp_{\mu}(s) = (\sum_{m \in mut(s)} \frac{|f_P(s) \cap p_m|}{\mathrm{f2p+1}} - \frac{|p_P(s) \cap f_m|}{\mathrm{p2f+1}})$ / ($|mut(s)|$ + 1)
    - $mut(s)$ is the set of all mutants of P that mutates s with observed changes in test results.
    - $f_P(s)$ and $p_P(s)$ are a set of failing tests and a set of passing tests that execute s on program target program P, respectively.
    - p_m and f_m are a set of failing and a set of passing tests on mutant m.
    - $f2p$ and $p2f$ are the number of test result changes from fail to pass and vice versa between before and after all mutants of P, the set of which is $mut(P)$.
  - $Susp_{\mathrm{MUSE}}(s) = Norm\_Susp(\mu, s) + Norm\_Susp(\mathrm{SBFL}, s)$
    - $Norm\_Susp(flt, s)$ is the normalized suspiciousness of a statement s in a fault localization technique flt, which is normalized into [0,1].
    - With this metric, we can give a meaningful suspiciousness to a statement s where $mut(s) = 0$.

MUSE: Example

int max; void Setmax(int x,int y){	Mutants	Test Result Changes					$\lvert f_P(s) \cap p_m \rvert$	$\lvert p_P(s) \cap f_m \rvert$	*MUSE*		*Jaccard*	
		ftc1 (3,1)	ftc2 (5,-4)	ptc3 (0,-4)	ptc4 (0,7)	ptc5 (-1,3)			Susp.	Rank	Susp.	Rank
1: max -= x; // **'max=x;'**	M1:max-=x-1;			P->F			0	1	1.40	1	0.40	5
	M2:max=x;	F->P	F->P				2	0				
2: if(max<y) {	M3:if(!(max<y)){			P->F	P->F	P->F	0	3	0.83	4	0.40	5
	M4:if(max==y){	F->P			P->F		1	1				
3: max = y;	M5:max-=y;				P->F	P->F	0	2	1.07	3	0.50	2
	M6:max=y+1;				P->F	P->F	0	2				
4: if(x*y<0)	M7:if(!(x*y<0))				P->F	P->F	0	2	1.14	2	0.50	2
	M8:if(x/y<0)					P->F	0	2				
5: print("diff. sign");}	M9:return;					P->F	0	2	0.21	6	0.33	6
	M10:;					P->F	0	2				
6: print("%d", max); }	M11:printf("%d",0);}				P->F	P->F	0	2	0.40	5	0.50	5
	M12:;}			P->F	P->F	P->F	0	3				

- MUSE perfectly locates the faulty statement, whereas the SBFL technique Jaccard does not.

MUSE: Overall Procedure



Empirical Evaluation

- Experimentation
  - Research questions
    - **RQ1**. Are the conjectures of MUSE valid?
    - **RQ2**. How precise is MUSE, compared with the SBFL techniques?
      - We compared MUSE with Jaccard, Ochiai, Op2 which are the state-of-art SBFL techniques.
    - **RQ3**. How precise is MUSE with a subset of mutants utilized, compared with the SBFL techniques
  - Subjects
    - 51 faulty versions of 5 real-world programs (6000~ 13000 LOC) from the SIR benchmark.

Target program	# of faulty version used	Size (LOC)	$\lvert f_P \rvert$	$\lvert p_P \rvert$	Description
flex 2.4.7	13	12,423	15.9	24.4	Lexical analyzer generator
grep 2.2	2	12,653	91.0	98.5	Patter matcher
gzip 1.1.2	7	6,576	34.3	178.6	Compression utility
sed 1.18	5	11,990	43.4	235.0	Stream editor
space	24	9,129	22.8	130.2	ADL interpreter
Average	10.2	10,554.2	41.48	133.3	

  - Experiments took 19 hours with 25 machines equipped with Intel i5 3.6Ghz quad core CPU
    - On average 29.85 mutants are used for each executed statement.

Our Conjectures Are Valid

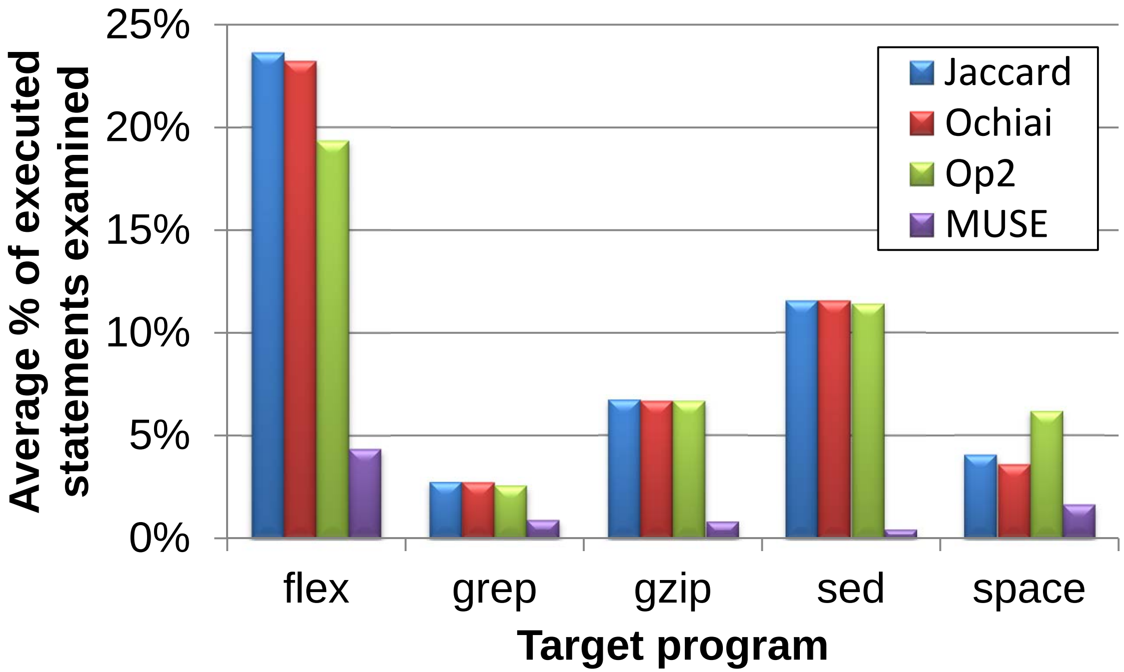
- **RQ1**. Are the conjectures of MUSE valid?
  - Conjecture Ⅰ, "mutating faulty statements is more likely to make originally failing tests pass than mutating correct statements", is **valid**.
  - Conjecture Ⅱ, "mutating correct statements is more likely to make originally passing tests fail than mutating faulty statements", is **valid**.

Target Program	# of failing tests that pass after mutating:			# of passing tests that fail after mutating:		
	faulty stmts. (A)	correct stmts. (B)	faulty/correct (A/B)	correct stmts. (C)	faulty stmts. (D)	correct/faulty (C/D)
flex	9.79	0.09	109.32	8.00	3.85	2.08
grep	38.69	8.31	4.66	13.27	3.22	4.11
gzip	3.68	0.10	35.29	87.80	4.13	21.25
sed	10.69	1.41	7.59	108.86	30.14	3.61
space	3.70	0.01	419.14	31.69	15.16	2.09
Average	13.31	1.98	115.20	49.92	11.30	6.63

➔ **We can expect that MUSE will localize faults precisely.**

MUSE Significantly Outperforms SBFL

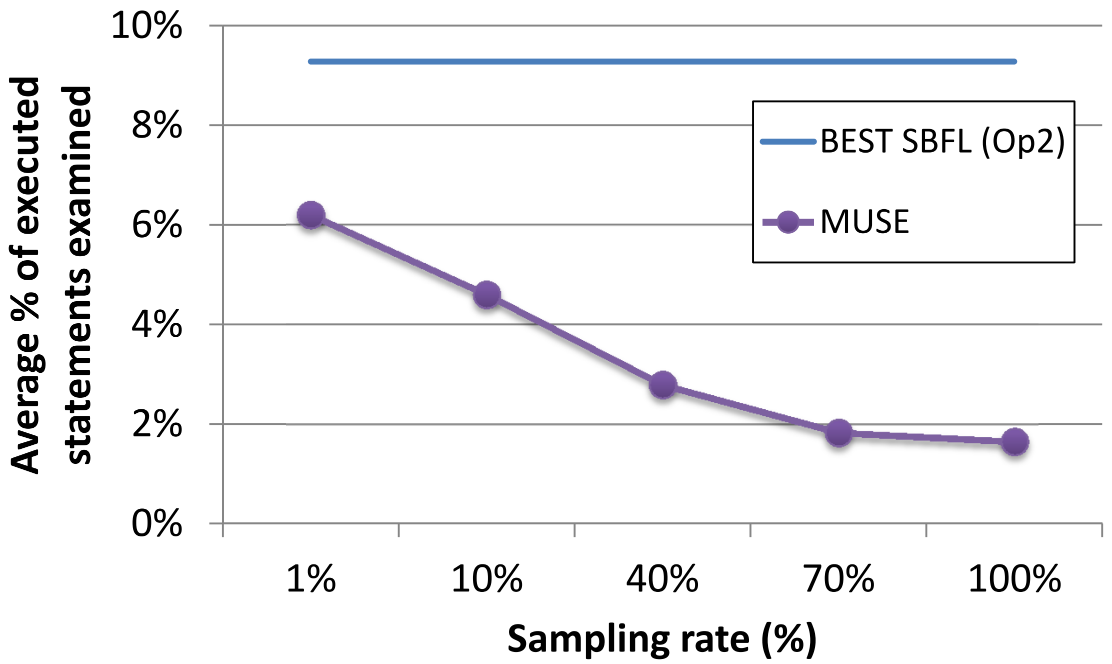
- **RQ2.** How precise is MUSE, compared with the SBFL techniques?



- On average, MUSE ranks a faulty statement top **1.65%** of executed statements.
  – The best-performing SBFL (i.e., Op2) ranks a faulty statement top **9.25%**.
- MUSE ranks a faulty statement among the top 10 for **38** faulty versions out of 51 faulty versions.
  – The best-performing SBFL (Op2) ranks a faulty statement among the top 10 for **9** faulty versions.

MUSE with Few Mutants Still Outperforms SBFL

- **RQ3.** How precise is MUSE with a subset of mutants utilized, compared with the SBFL techniques?



- MUSE with mutant sampling rate **1%** requires a developer to inspect **6.2%** of executed statements.
- MUSE with only 1% generated mutants shows better performance than the best SBFL technique.

Conclusion and Future Work

- MUSE is a new fault localization technique which is highly precise and widely applicable based on mutation analysis [ICST'14].
- Future work
  - User study and more empirical study to show that MUSE actually helps developers locate faults quickly
  - Additional techniques to improve fault localization
    - Automatic test case generation for enhancing fault-localization
    - Clustering highly suspicious target statements to speed up the review process
    - Backward/forward iterative symbolic analysis to narrow down candidate faulty statements
  - Applying MUSE to very large size real-world programs including real-faults (e.g., PHP (1MLOC)).
    - For randomly selected 10 PHP faults among the PHP bugs used by GenProg (ICSE2012),
      - Faulty stmt rank: MUSE **25.3** / SBFL (Op2): 84.2
        - » For each faulty version, we randomly selected 100 passing test cases from all test cases that execute at least one line of faulty file.