ASK THE MUTANTS

MUTATING FAULTY PROGRAMS FOR FAULT LOCALISATION

Shin Yoo, University College London

Seokhyun Mun, KAIST

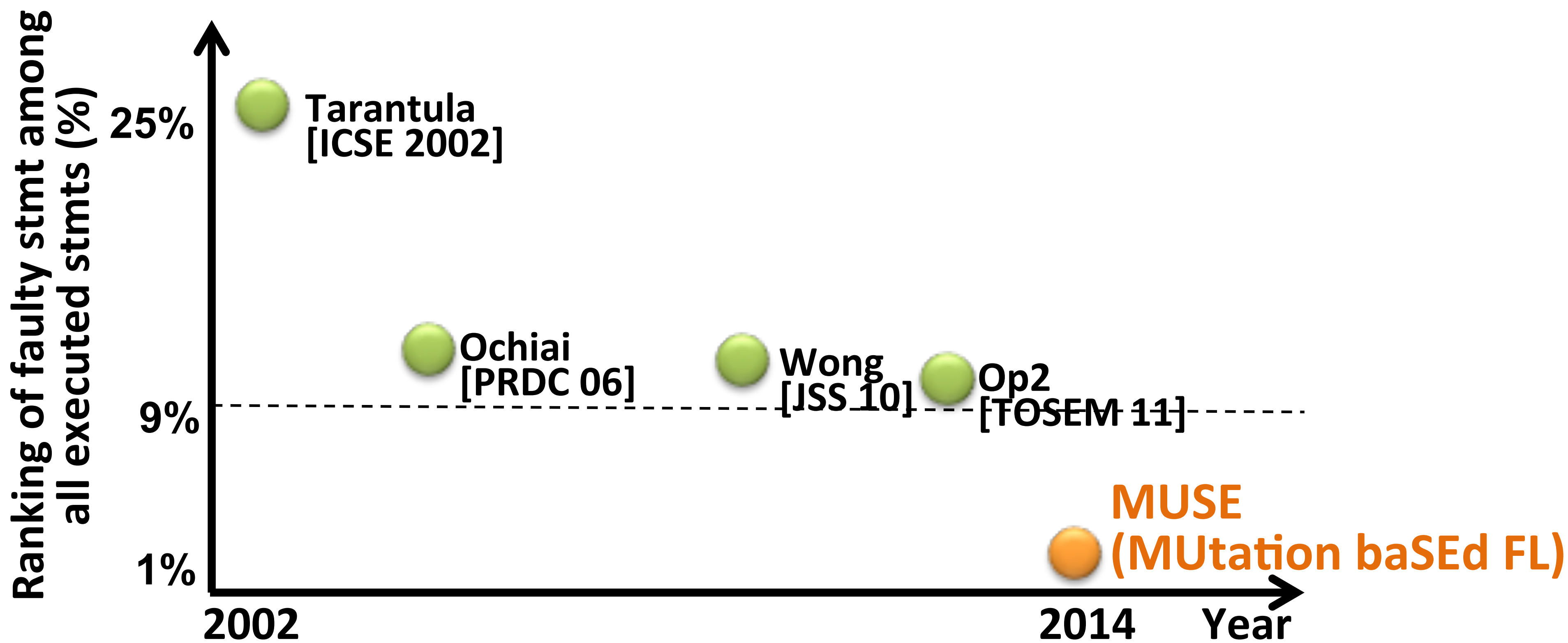
Yunho Kim, KAIST

Moonzoo Kim, KAIST

OUTLINE

- MUSE: Mutation-based Fault Localisation Engine
- Locality Information Loss: a new evaluation metric
- Ongoing work (post ICST 2014)

OUR CLAIM



On the 10KLOC SIR benchmark programs

MOTIVATION

- We have hit the ceiling of Spectrum-based Fault Localisation
  - Not accurate enough, effectiveness varies significantly depending on test suites, inherently limited by block-level granularity
- Can we use mutation testing in a pre-emptive manner?

WHAT HAPPENS WHEN YOU MUTATE ALREADY FAULTY PROGRAMS?

P F

CASE 1: MUTATING CORRECT STATEMENTS

New Fault

P- F+

Equivalent

P F

Mask

P+ F-

CASE 2: MUTATING FAULTY STATEMENT

(Partial) Fix

P+ F-

(New) Fault

P? F?

Equivalent

P F

Mask

P+ F-

HYPOTHESES

- An arbitrary mutation operator applied to a correct statement is likely to introduce a new fault
- An arbitrary mutation operator applied to a faulty statement is either likely to keep the program still faulty or, even better, (partially) fix the program
- The majority of statements in a faulty statement is correct; we detect the faulty one by observing the anomaly from our hypotheses

MUSE

Proportion of test cases that mutant m turns from fail to pass

$$\mu(s) = \frac{1}{|mut(s)|} \sum_{m \in mut(s)} \left(\frac{|f_P(s) \cap p_m|}{|f_P|} - \alpha \cdot \frac{|p_P(s) \cap f_m|}{|p_P|} \right)$$

Average over all mutation applied to statement s

Proportion of test cases that mutant m turns from pass to fail

$$\alpha = \frac{f2p}{|mut(P)| \cdot |f_P|} \cdot \frac{|mut(P)| \cdot |p_P|}{p2f}$$

EMPIRICAL EVALUATION

Subject Program	% of executed stmts examined				Rank of a faulty stmt			
	Jaccard	Ochiai	Op2	MUSE	Jaccard	Ochiai	Op2	MUSE
`flex v1`	49.48	45.04	32.01	**0.04**	1,371	1,248	887	**1**
`flex v7`	3.60	3.60	3.60	**0.07**	100	100	100	**2**
`flex v11`	19.76	19.54	13.51	**0.04**	547	541	374	**1**
`grep v3`	1.06	1.01	**0.71**	1.87	21	20	**14**	37
`grep v11`	3.44	3.44	3.44	**1.60**	58	58	58	**27**
`gzip v2`	2.14	2.14	2.14	**0.07**	31	31	31	**1**
`gzip v5`	1.83	1.83	1.83	**0.07**	26	26	26	**1**
`gzip v13`	1.03	1.03	1.03	**0.07**	15	15	15	**1**
`sed v1`	**0.54**	**0.54**	**0.54**	0.90	**12**	**12**	**12**	20
`sed v3`	2.56	2.56	2.56	**0.13**	57	57	57	**3**
`sed v5`	37.84	37.84	37.15	**0.28**	814	814	799	**6**
`space v19`	**0.03**	**0.03**	**0.03**	0.06	**1**	**1**	**1**	2
`space v21`	0.45	0.45	0.45	**0.03**	15	15	15	**1**
`space v28`	11.57	10.66	6.89	**0.04**	329	303	196	**1**
Average	9.67	9.27	7.56	**0.38**	242.64	231.50	184.64	**7.43**

MOTIVATION

- Traditional evaluation metric for fault localisation is ranking based
  - Measures something else than accuracy (and, even then, humans do not operate in linear ranking)
  - Irrelevant for Automated Patching: Qi et al. show that rank-wise suboptimal formula helps GenProg better (ISSTA 2013)

LIL(LOCALITY INFORMATION LOSS)

- Any suspiciousness score distribution can be interpreted as a probability distribution
- Describe the actual location of the fault as THE probability distribution
- Calculate Kullbeck-Leibler divergence (distance between two probability distributions)

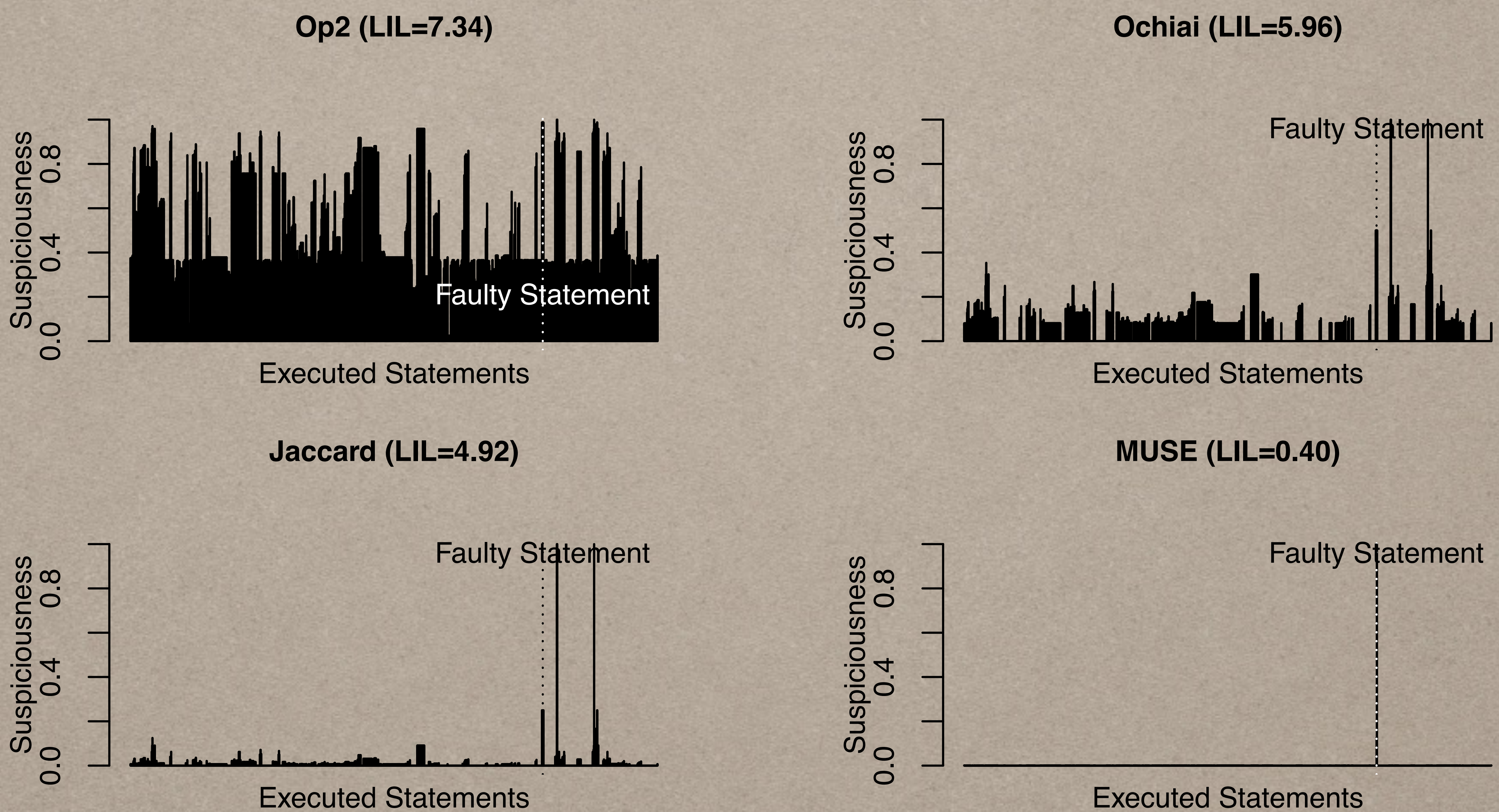
LIL

$$\mathcal{L}(s_i) = \begin{cases} 1 & (s_i = s_f) \\ \epsilon & (0 < \epsilon \ll 1, s_i \in S, s_i \neq s_f) \end{cases}$$

$$P_\tau(s_i) = \frac{\tau(s_i)}{\sum_{i=1}^{n} \tau(s_i)}, (1 \leq i \leq n)$$

$$D_{KL}(P_{\mathcal{L}} || P_\tau) = \sum_i \ln \frac{P_{\mathcal{L}}(s_i)}{P_\tau(s_i)} P_{\mathcal{L}}(s_i)$$

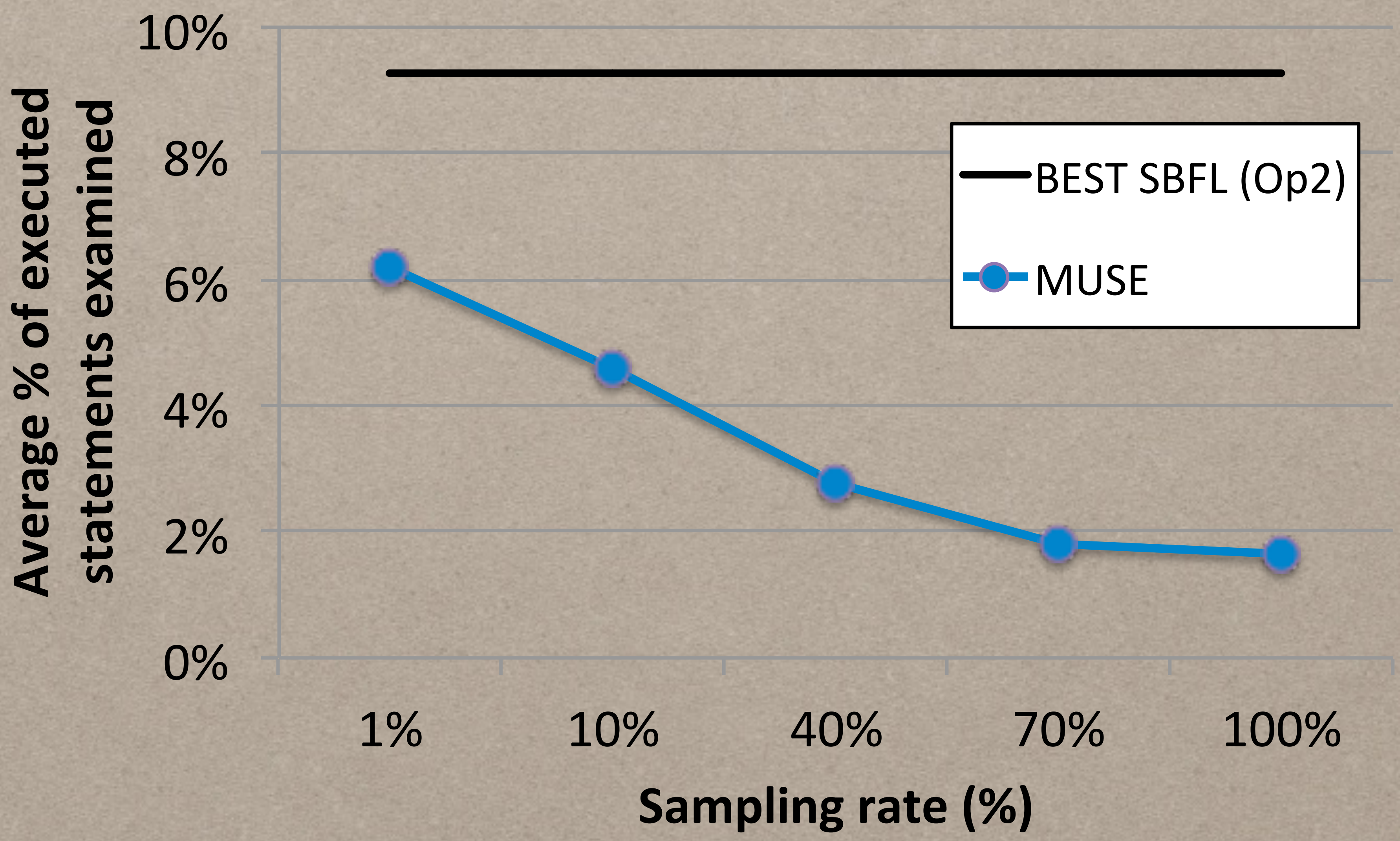
..WORTH A THOUSAND WORDS



ONGOING WORK

- With actual progress
  - Mutant Sampling
  - Hybridisation
- More adventurous
  - Multiple Faults
  - Pre-emptive localisation
  - Learning mutation
  - Genetic Programming for MUSE

ISN'T MUTATING EVERY STATEMENT REALLY EXPENSIVE?



After random sampling, still more accurate than Op2

MUSE + SBFL(JACCARD)

- Evaluated 10 randomly chosen real PHP bugs from ICSE 2012 GenProg Paper
  - In some cases, brings MUSE closer to Op2
  - In other cases, pure version still wins

Faulty version #	MUSE	MUSE +jacc (Current Hybrid Muse)	Op2	Target line #	# failed	# passed
1	4	**1**	160	1957	1	100
2	**6**	13	53	112	1	100
3	42	**14**	**14**	143	2	100
4	177	**8**	**8**	362	1	100
5	371	**140**	**140**	553	1	100
6	**4**	**4**	170	191	1	100
7	**5**	18	64	119	1	100
8	**1**	9	164	222	1	100
9	49	**45**	**45**	49	2	100
10	**1**	**1**	24	362	1	100
Average	66	25.3	84.2	407.00	1.20	100.00

MULTIPLE FAULTS CONJECTURE

- For independent faults that results in disjoint failures, MUSE is not affected at all
- For faults that interact with each other, test suite design/composition will play a key role

PRE-EMPTIVE LOCALISATION

- Mutation analysis is still expensive, especially as a step for debugging which is often urgent
- Can we do the mutation analysis in advance, even with the previous version?
  - For each mutant, record the failure pattern across test cases
  - When a real failure is observed, track back to the point of mutation

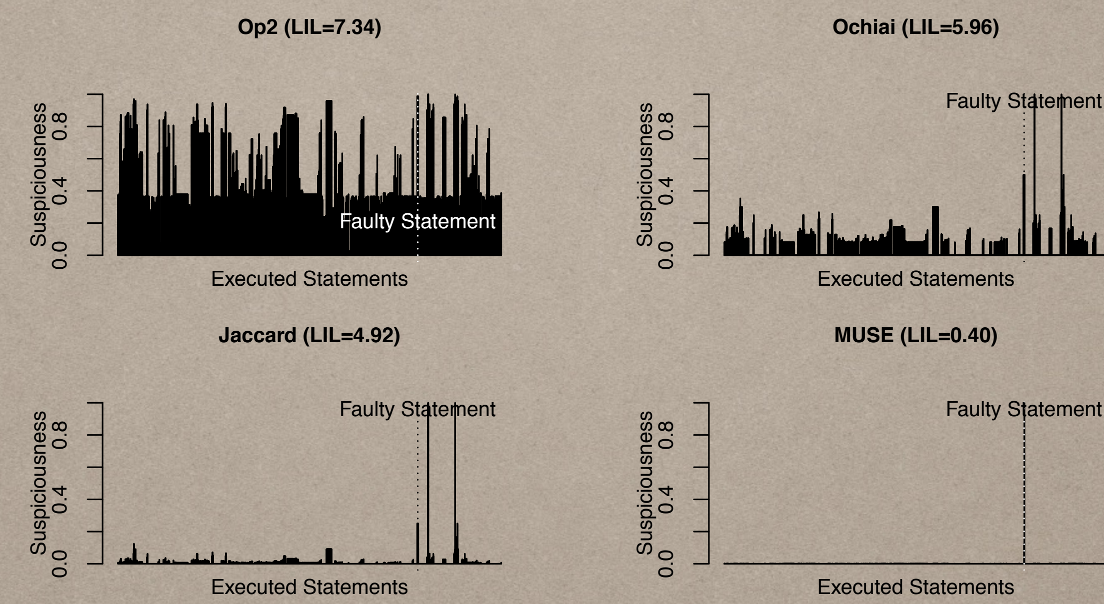
GP FOR MUSE

- GP worked for SBFL; does it work for MUSE?
- More variables, which means a larger search space

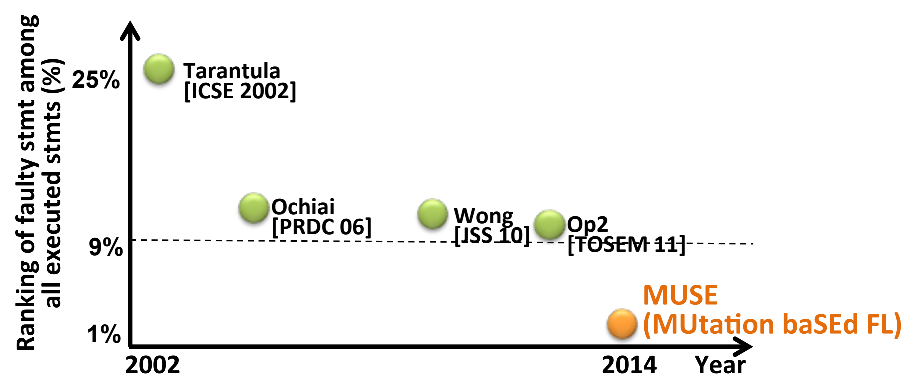
WHAT HAPPENS WHEN YOU MUTATE ALREADY FAULTY PROGRAMS?

P F

..WORTH A THOUSAND WORDS

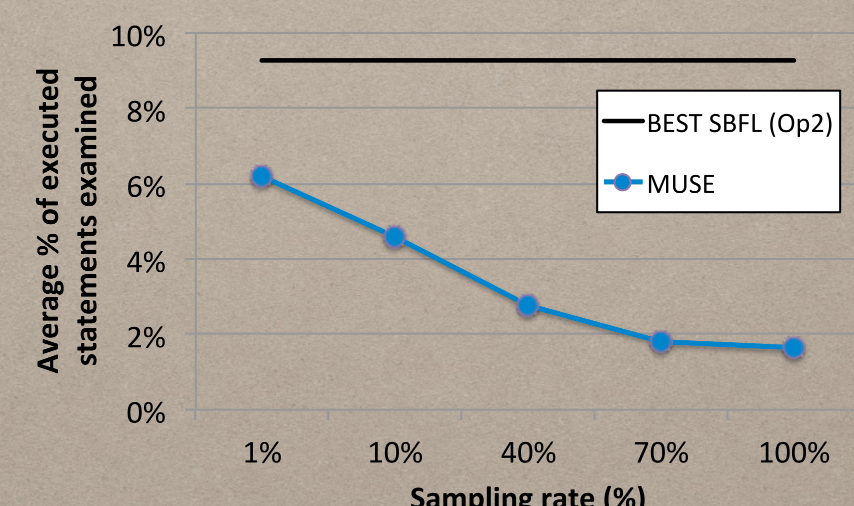


OUR CLAIM



On the 10KLOC SIR benchmark programs

ISN'T MUTATING EVERY STATEMENT REALLY EXPENSIVE?



After random sampling, still more accurate than Op2