Evaluating Directed Fuzzers: Are We Heading in the Right Direction?

Tae Eun Kim, Jaeseung Choi, Seongjae Im, Kihong Heo, Sang Kil Cha

KAIST

SOGANG UNIVERSITY

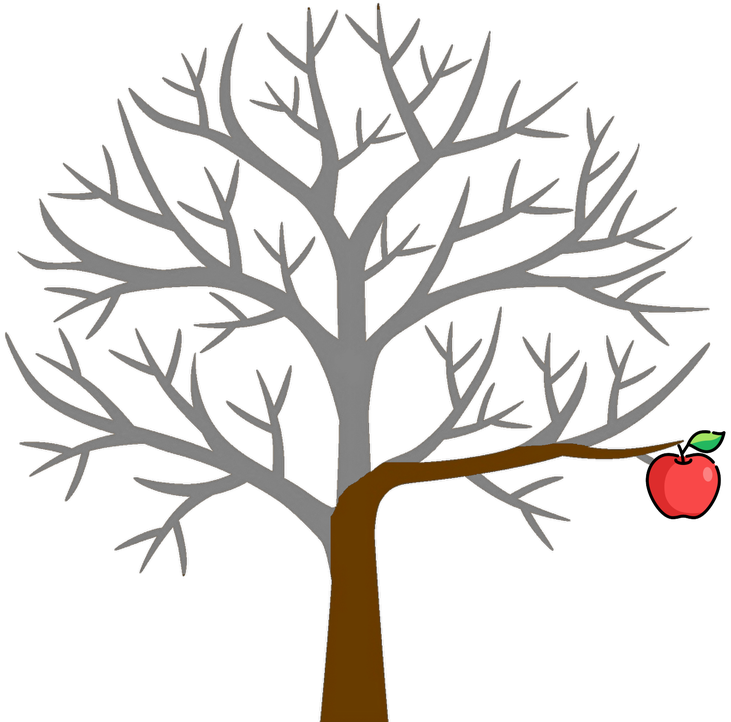
Background

Fuzzing

- Testing a program with randomly generated inputs
- Successful achievements
  - e.g., AFL, Google's OSS Fuzz project

Directed Fuzzing

- Aims to test a specific part of the program
  - e.g., generate crashing inputs from bug reports



Background

Evaluation of Directed Fuzzing

Key metric: How fast does it expose a given target bug?

→ Time-To-Exposure (TTE)

Problem:

- No standards in the directed fuzzing evaluation
- Pitfalls specific to directed fuzzing are often overlooked

→ An obstacle to the transparency and reproducibility of the evaluation

Pitfalls of Evaluating Directed Fuzzers

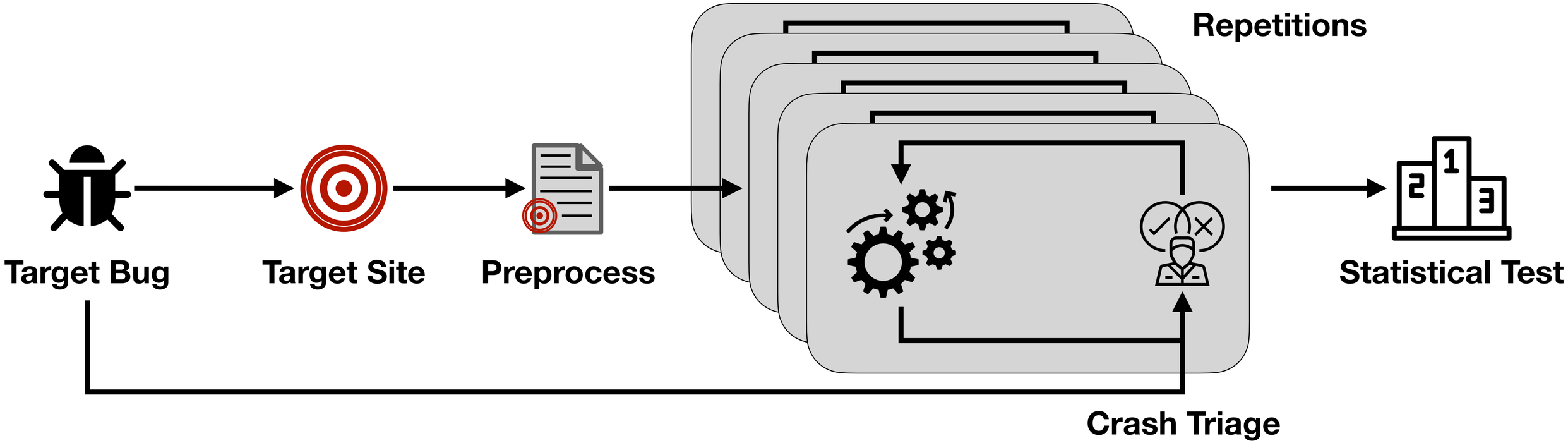
Survey: Evaluation process of 14 directed fuzzing papers

Experiment: 5 state-of-the-art directed fuzzers on 12 widely used benchmarks

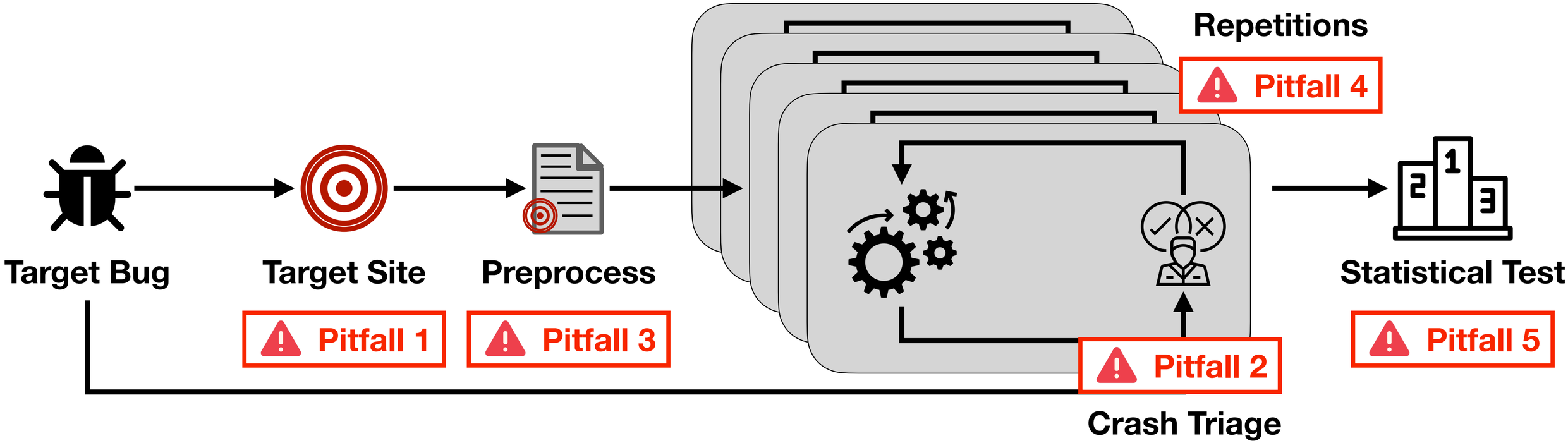
Findings:

- 5 pitfalls in each step of the evaluation process
- 5 lessons for transparent and reproducible evaluations

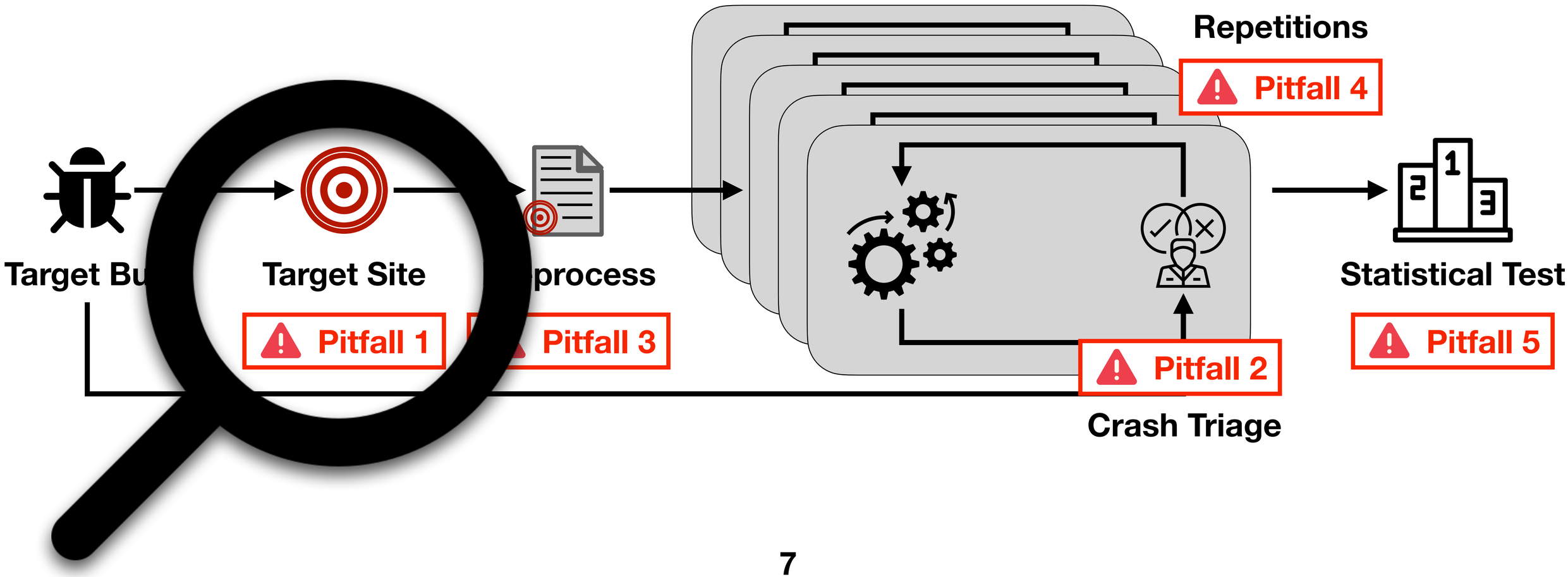
Process of Directed Fuzzing Evaluation



Process of Directed Fuzzing Evaluation



Pitfall 1: Target Site



Pitfall 1: Target Site

Target site selection from the given target bug is complicated

Current Practice: Most papers specify target bugs with *CVE IDs (12 out of 14)

Problem:

- Target bug is the goal of the ***evaluation***, not the goal of the ***directed fuzzer***
- Most directed fuzzers take target line as an input, instead of target bug

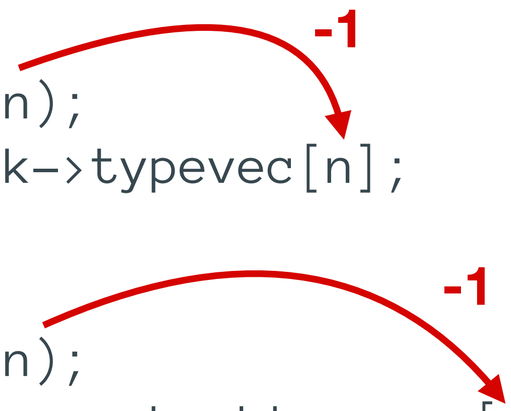
→ Such discrepancy may cause inconsistent results

*Common Vulnerabilities and Exposure

Pitfall 1: Target Site

Ex) *CVE-2016-4492: Bug with two crashing sites

```
1  int do_type(work_stuff *work, char **mangled)
2    int n;
3    switch (**mangled) {
4      case 'T':
5        get_count (mangled, &n);
6        remembered_type = work->typevec[n];          // Crash Site 1
7        ...
8      case 'B':
9        get_count (mangled, &n);
10       string_append (result, work->btypevec[n]); // Crash Site 2
11   }
12 }
```



*Used in 6 out of 14 papers

Pitfall 1: Target Site

Q. Why not choose any line?

A. The results differ significantly

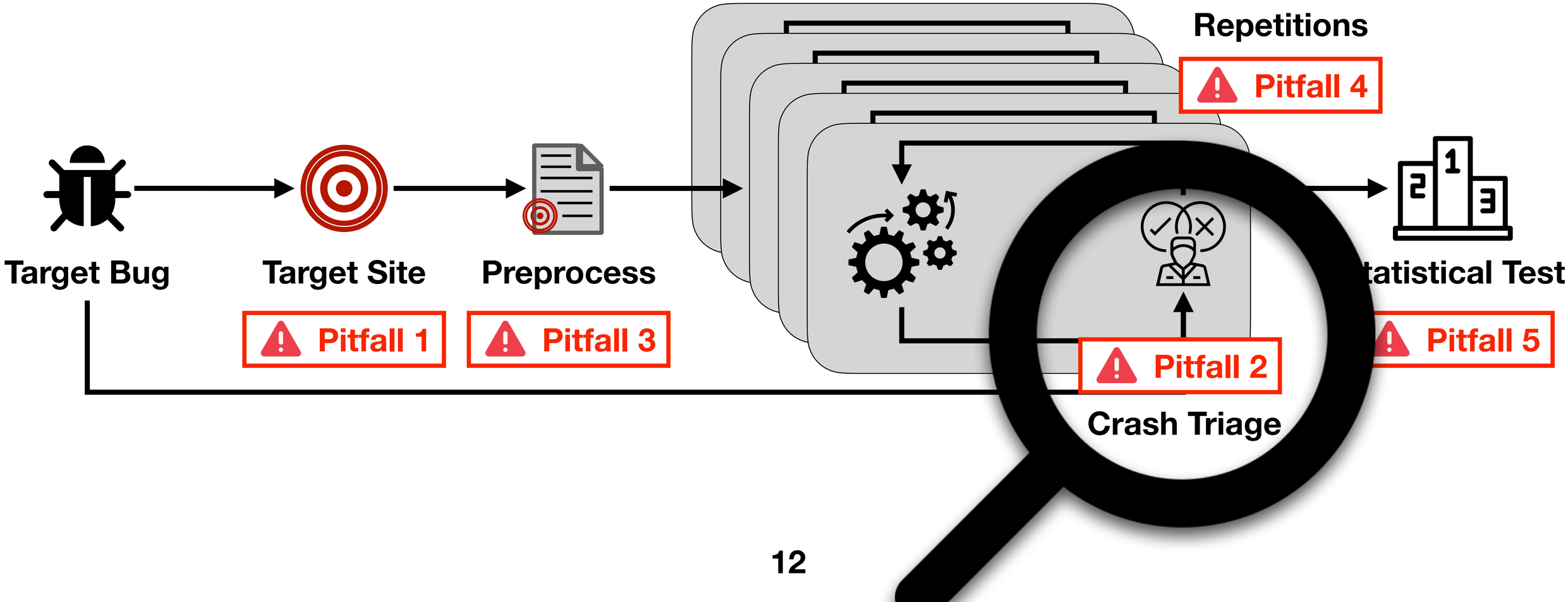
*** Median TTE of 160 repetitions in seconds**

Target Line	AFLGo	Beacon	WindRanger	SelectFuzz	DAFL
Line 6	373	333	2,460	432	787
Line 10	332	499	339	581	149

Pitfall 1: Target Site

- ⚠️ 6 out of 12 papers report only the CVE IDs
- ✅ Report the exact target line provided to the directed fuzzers

Pitfall 2: Crash Triage



Pitfall 2: Crash Triage

```
ERROR: AddressSanitizer: heap-buffer-overflow …
  #0 in parseSWF_RGBA parser.c:66
  #1 in parseSWF_MORPHGRADIENTRECORD parser.c:746
     ...
  #6 in blockParse blocktypes.c:145
  #7 in readMovie main.c:265
  #8 in main main.c:350
```

TTE is dependent on the details of the triage logic

Current Practice: Sanitizer-based triage

- Utilizing sanitizer logs such as ASAN reports (crash type, stack trace)
- Compare the found crashing input with
  - Description of the CVE
  - Sanitizer log of the *POC input provided in the CVE report

Problem: Deciding the details of the comparison is not trivial

*Proof of Concept

Pitfall 2: Crash Triage

Ex) CVE-2016-9831

CVE report:

"*Heap-based buffer overflow in the* `parseSWF_RGBA` *function*"

```
1  void parseSWF_MORPHGRAD(FILE *f,
2    ...
3    g->NumGradients = readUInt8(f);   <------- NumGradients is not validated
4    for (i = 0; i < g->NumGradients; i++)
5      parseSWF_MORPHGRADREC(f, &(g->GradientRecords[i]));
6  }
7
8  void parseSWF_MORPHGRADREC(FILE *f, SWF_MORPHGRADREC *r) {
9    r->StartRatio = readUInt8(f);   <-------- Same bug can also crash here
10   parseSWF_RGBA(f, &r->StartColor);
11 }
12
13 void parseSWF_RGBA(FILE *f, SWF_RGBA *rgb) {
14   rgb->red    = readUInt8(f);   <------ POC in the CVE report crashes here
15   rgb->green  = readUInt8(f);   <-------- CVE report mentions this line too
16 }
```

Pitfall 2: Crash Triage

Ex) CVE-2016-9831

Lines Checked	AFLGo	Beacon	WindRanger	SelectFuzz	DAFL
14	1,418	1,069	487	1,777	1,218
14,15	167	177	174	218	103
14,15, 9	159	155	155	200	93

```
 9    r->StartRatio = readUInt8(f);
10    parseSWF_RGBA(f, &r->StartColor);
11  }
12
13  void parseSWF_RGBA(FILE *f, SWF_RGBA *rgb) {
14    rgb->red    = readUInt8(f);
15    rgb->green  = readUInt8(f);
16  }
```

Same bug can also crash here

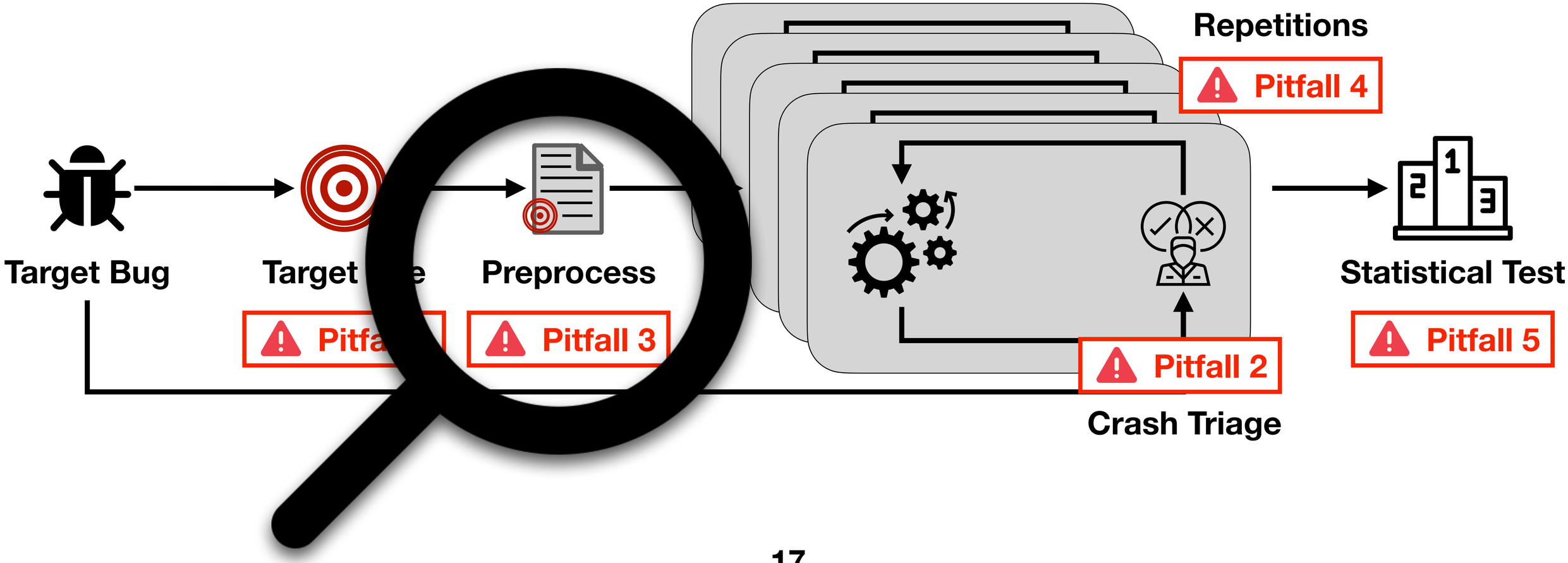
POC in the CVE report crashes here

CVE report mentions this line too

Pitfall 2: Crash Triage

- ⚠️ Only 5 papers disclose the details of the triage logic
- ✅ Clearly specify crash triage logic and disclose its code

Pitfall 3: Preprocessing



Pitfall 3: Preprocessing

Omitting preprocessing time can be misleading

Current Practice: Most directed fuzzers utilize static analysis (12 out of 14)

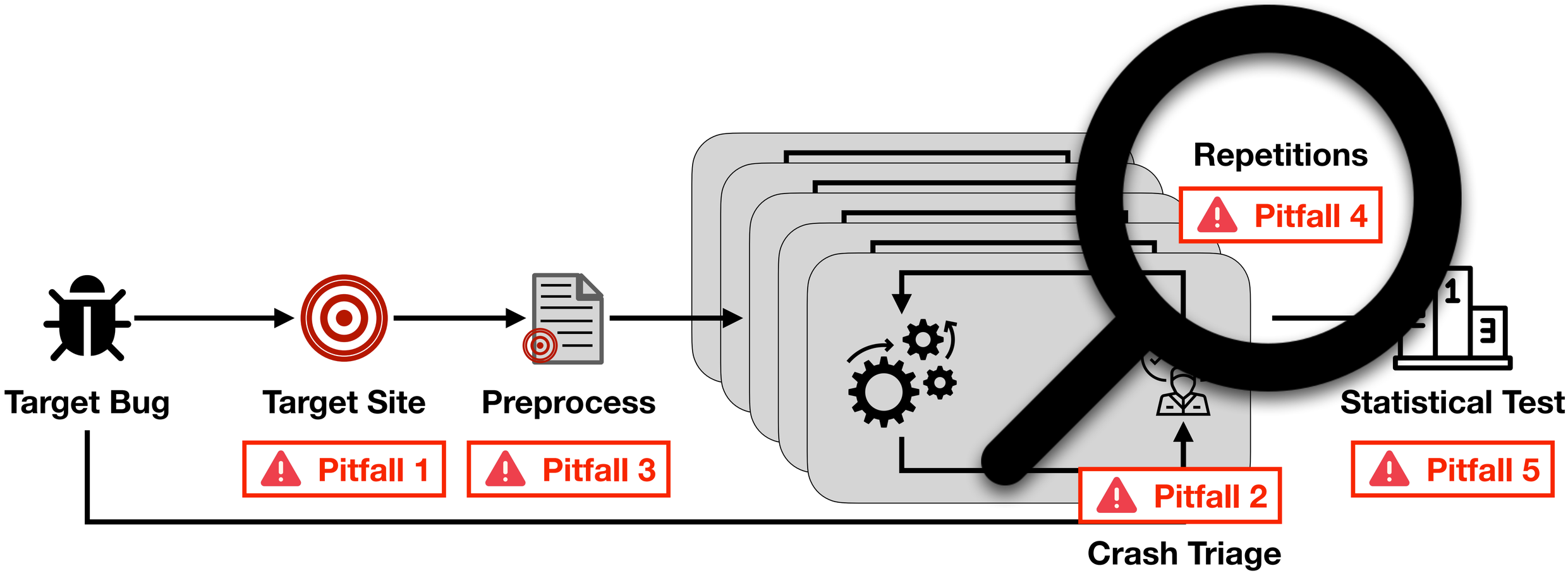
Problem:

- Static analysis time is often not a one-time cost
- Static analysis time can be greater than the fuzzing time

⚠️ Only 3 papers fully disclose the static analysis time

✅ Report end-to-end time of evaluation to better understand the performance

Pitfall 4: Repetitions



Pitfall 4: Repetitions

Randomness has severe impact in directed fuzzing

Regular Fuzzing: Measures the coverage rate or the number of found bugs

Directed Fuzzing: Measures the found time of a specific target bug

Current Practice: All papers repeat experiments multiple times

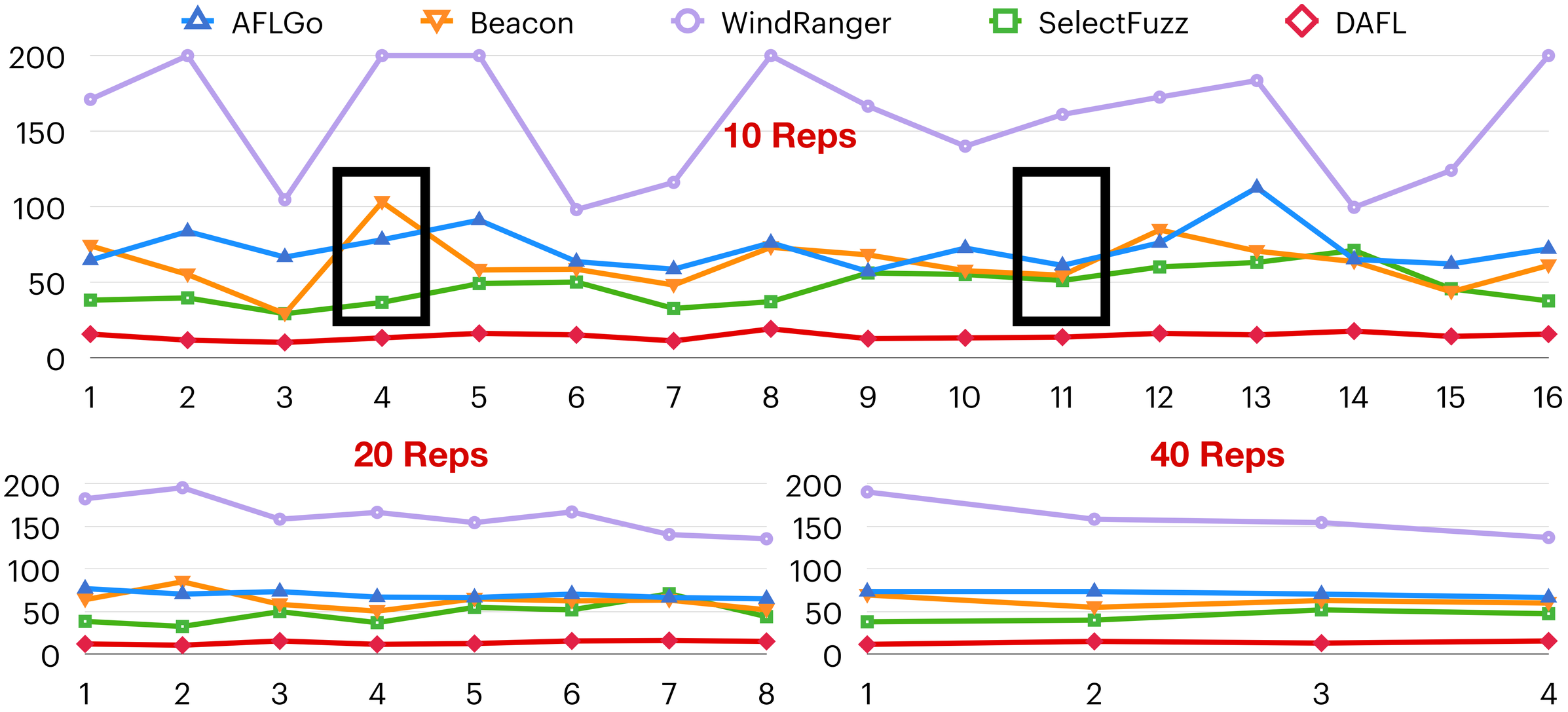
Problem: The number of repetitions is often not enough

Pitfall 4: Repetitions

Ex) CVE-2016-4490: Moderate case without timeouts

- Repeated 160 times, grouped by 10, 20, and 40 repetitions
- Compared the median TTE of each groups

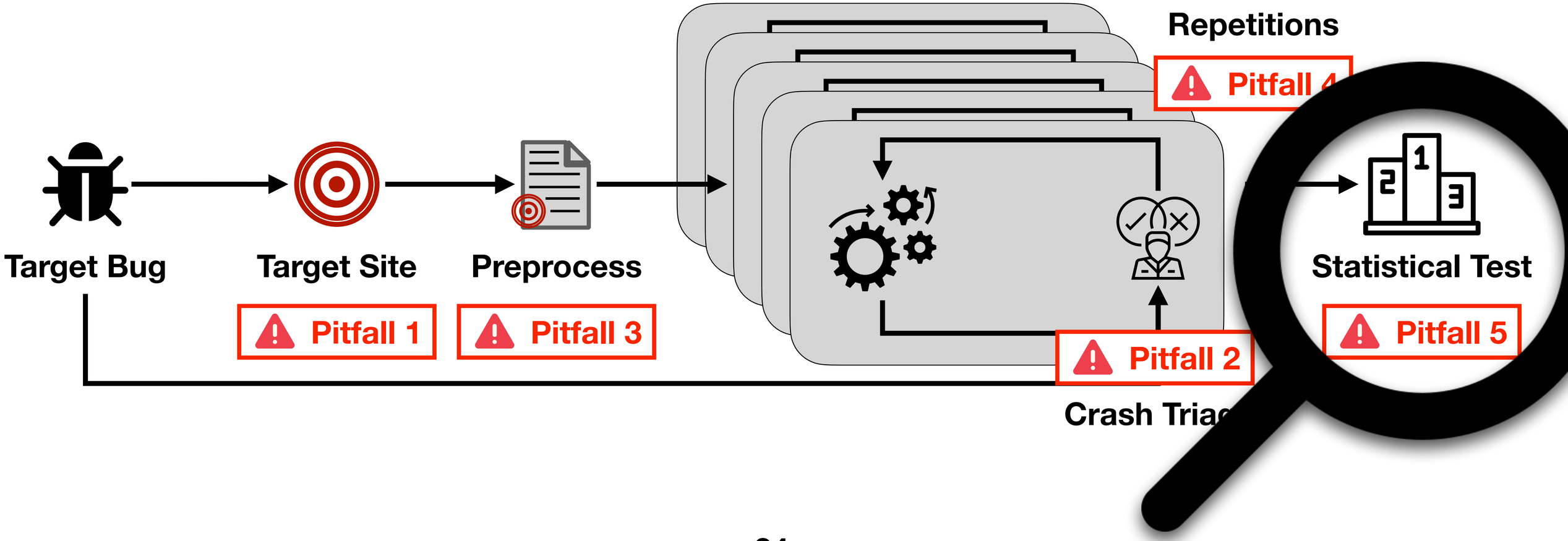
Pitfall 4: Repetitions



Pitfall 4: Repetitions

- ⚠️ The number of repetition is 16 on average, 10 or less for half of the papers
- ✅ Repeat at least 20 times or more

Pitfall 5: Statistical Testing



Pitfall 5: Statistical Testing

Usage of inappropriate statistical test can mislead the conclusion

Current Practice:

Utilize the Mann-Whitney U (MWU) test to check the significance of the result

Problem: MWU cannot handle data from “unobserved” events (e.g., Timeouts)

- Choice 1: Provide the time limit as TTE ⇢ Imprecise
- Choice 2: Eliminate timeout cases from the result ⇢ Biased

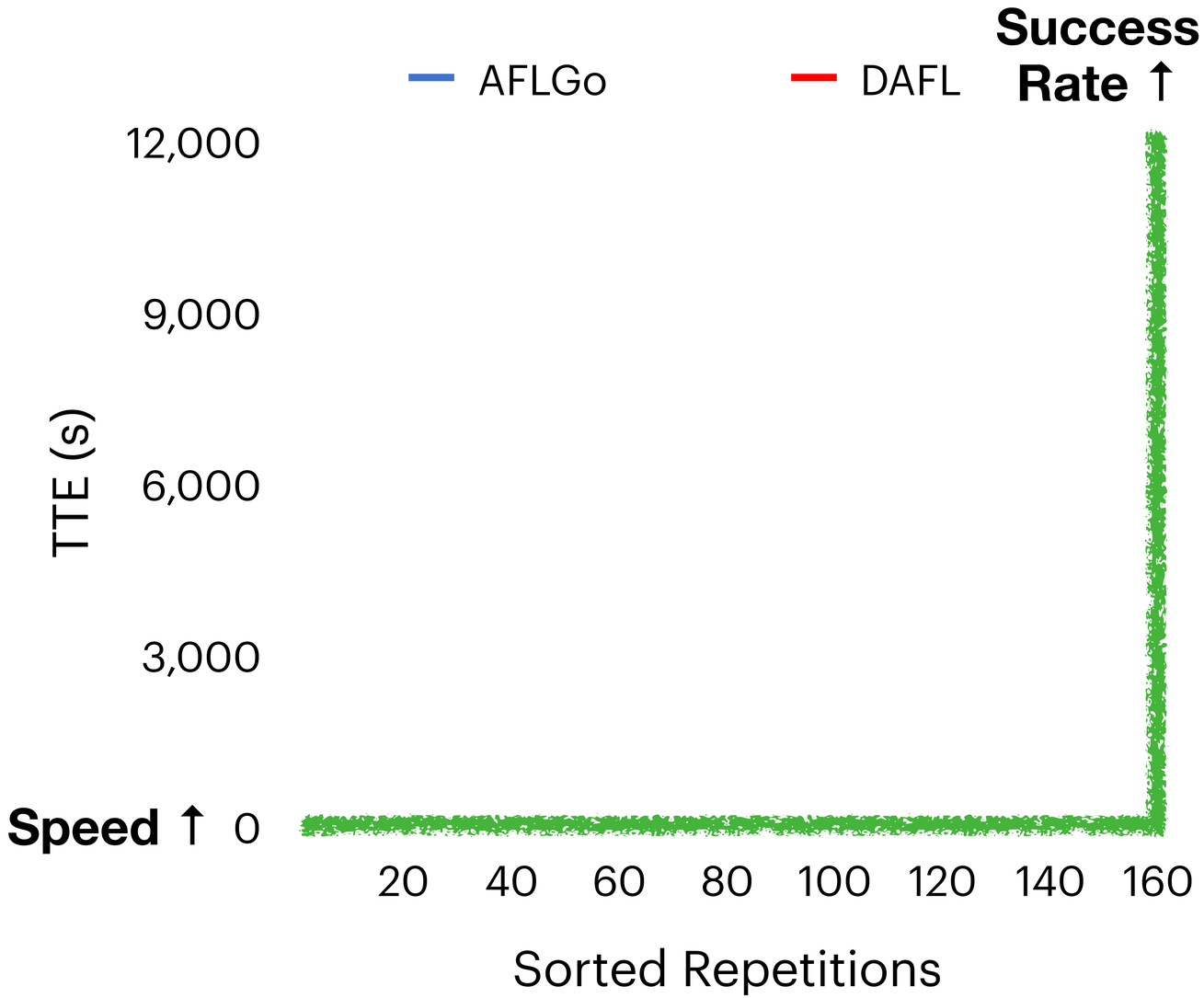
Pitfall 5: Statistical Testing

Ex) CVE-2017-9988

Statistics	AFLGo	DAFL
Median TTE	1,066	703
MWU test	p-value < 0.05	
# Timeouts	1	17
Logrank test	p-value > 0.5	

* **p-value:** A statistical test result is considered to be significant if the p-value is less than 0.05

* **Logrank test:** Statistical test used in survival analysis. Correctly handles timeout cases.



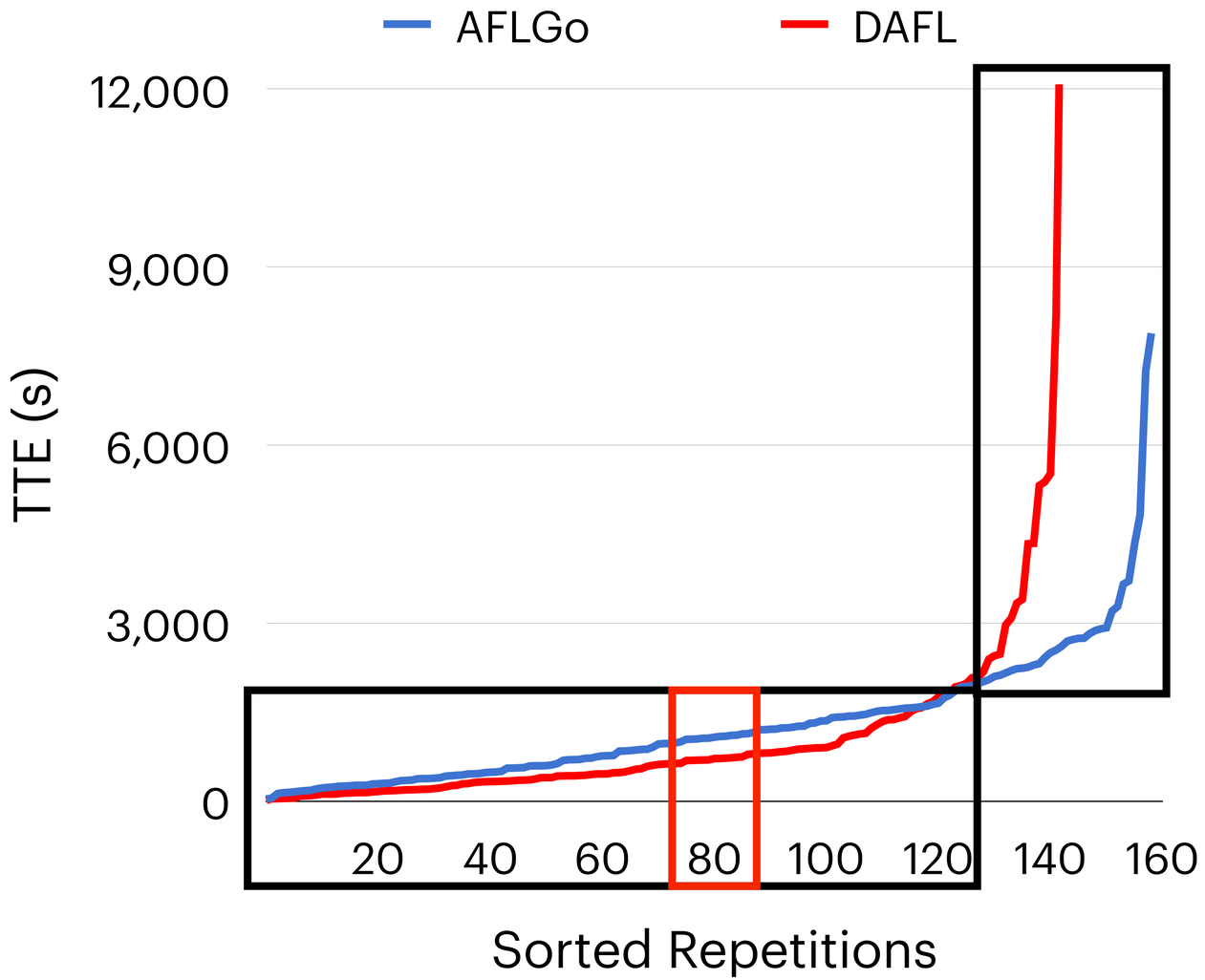
Pitfall 5: Statistical Testing

Ex) CVE-2017-9988

Statistics	AFLGo	DAFL
Median TTE	1,066	703
MWU test	p-value < 0.05	
# Timeouts	1	17
Logrank test	p-value > 0.5	

* **p-value:** A statistical test result is considered to be significant if the p-value is less than 0.05

* **Logrank test:** Statistical test used in survival analysis. Correctly handles timeout cases.



Pitfall 5: Statistical Testing

- ⚠️ 8 papers rely on the MWU test
- ✅ Use the Logrank test and cactus plot rather than the MWU test

Summary

Artifact Link

Lessons for evaluation of directed fuzzing

- Report the exact target line provided to the directed fuzzers
- Specify crash triage logic and disclose its code
- Report end-to-end time of evaluation including the preprocessing time
- Repeat at least 20 times or more to mitigate randomness
- Use the Logrank test and cactus plot rather than the MWU test

More details in the Paper!