MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

MONASH University

On the Surprising Efficiency and Exponential Cost of Fuzzing

Marcel Böhme
Software Security
MPI-SP & Monash

Keywords: Vulnerability Discovery, Automated Software Testing, Effectiveness, Efficiency, Scalability, Guarantees

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

MONASH University

On the Surprising Efficiency and Exponential Cost of Fuzzing

We might have strong intuitions about a problem,
but without a deep understanding of the problem
our intuitions might lead us astray.

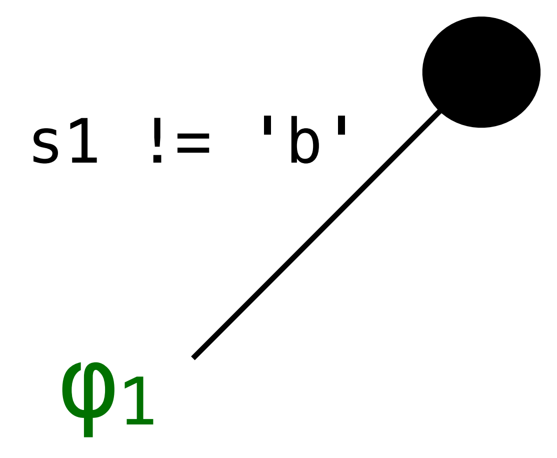
Marcel Böhme
Software Security
MPI-SP & Monash

Keywords: Vulnerability Discovery,
Automated Software Testing,
Effectiveness, Efficiency,
Scalability, Guarantees

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

Whitebox Fuzzing

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

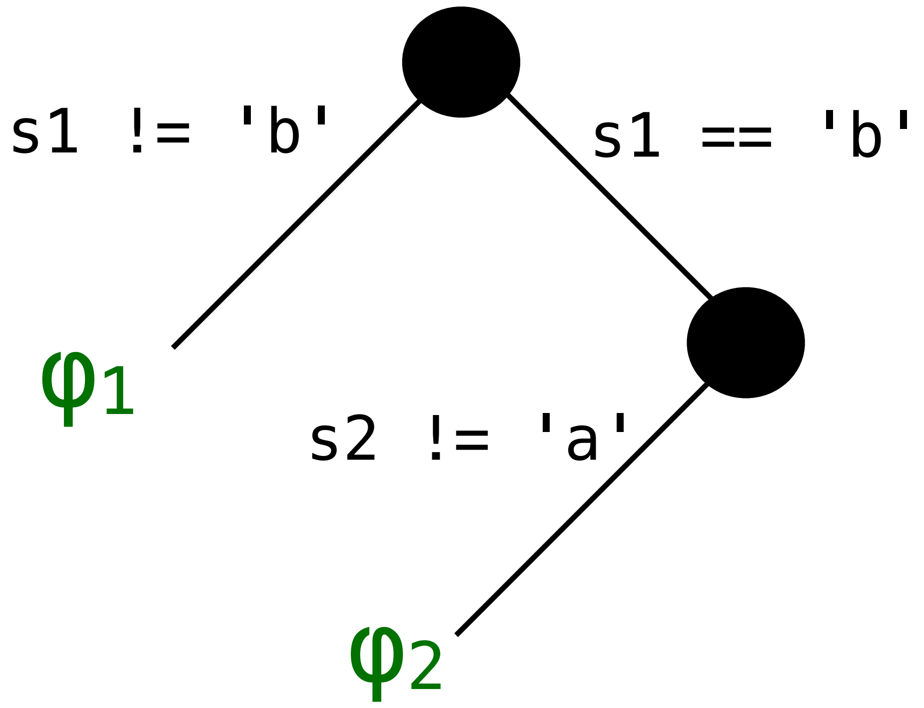


Path Conditions

✓ φ_1 = (s0 != 'b')

Whitebox Fuzzing

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```



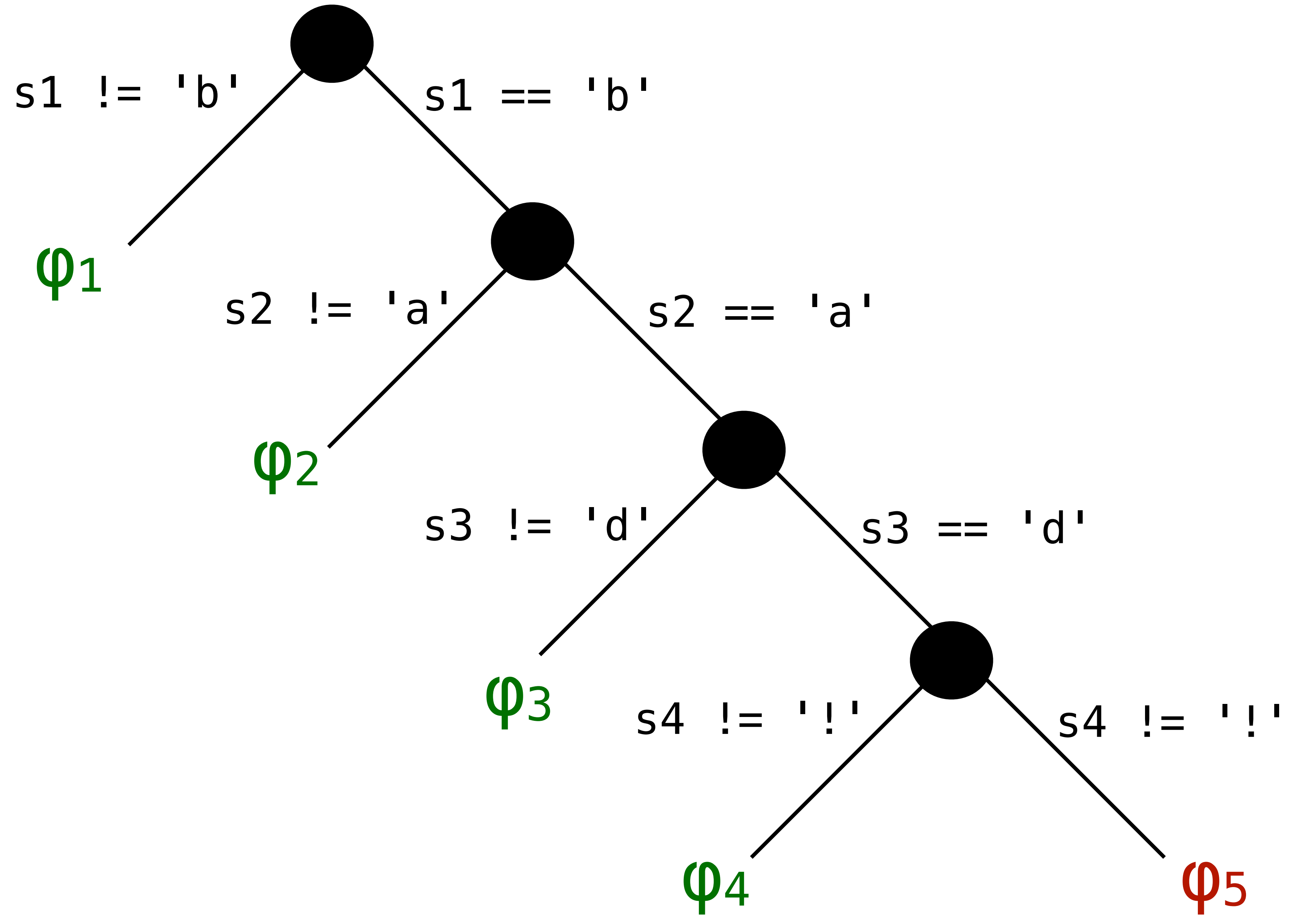
Path Conditions

✓ φ1 = (s0 != 'b')

✓ φ2 = (s0 == 'b') /\ (s1 != 'a')

Whitebox Fuzzing

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```



Path Conditions

✓ φ_1 = (s0 != 'b')

✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')

✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')

✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')

✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Whitebox Fuzzing: Most Effective!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

Path Conditions

✓ φ1 = (s0 != 'b')

✓ φ2 = (s0 == 'b') /\ (s1 != 'a')

✓ φ3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')

✓ φ4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')

✗ φ5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Whitebox Fuzzing: Most Effective!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

← It can prove the absence of assertion violation, by enumerating all paths and modulo some assumptions.

Path Conditions

✓ φ_1 = (s0 != 'b')

✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')

✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')

✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')

✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Whitebox Fuzzing: Quite Efficient!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

Path Conditions

✓ φ_1 = (s0 != 'b')

✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')

✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')

✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')

✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Whitebox Fuzzing: Quite Efficient!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

We only need 3 inputs to find the bug, on average, if we choose each path at random without replacement.

Choose a random path from the multivariate hypergeometric (i.e., enumerate).
Choose some input that exercises that path (by constraint solving).

Path Conditions

✓ φ_1 = (s0 != 'b')

✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')

✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')

✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')

✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

← It can never prove the absence of assertion violation!

August 1969

NOTES ON STRUCTURED PROGRAMMING by prof.dr.Edsger W.Dijkstra

On the reliability of mechanisms.

Corollary of the first part of this section:

Program testing can be used to show the presence of bugs, but never to show their absence!

https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

← It can never prove the absence of assertion violation!

Well, that's not entirely true. We can estimate a "residual risk".

August 1969

NOTES ON STRUCTURED PROGRAMMING by prof.dr.Edsger W.Dijkstra

On the reliability of mechanisms.

Corollary of the first part of this section:

Program testing can be used to show the presence of bugs, but never to show their absence!

[ESEC/FSE'21] **Estimating Residual Risk in Greybox Fuzzing,** M Böhme, D Liyanage, V Wüstholz

[TOSEM'18] **STADS: Software Testing as Species Discovery,** M Böhme; ACM Trans. Softw. Eng. Meth.

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.

So, whitebox fuzzing is better, right?

Blackbox Fuzzing: just random, really.

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

For each parameter, choose 1 of 256 values uniformly at random.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1}$ ≈ **4 billion inputs**, in expectation.

So, whitebox fuzzing is better, right? Wrong. At least not always.

Partition Testing Does Not Inspire Confidence

Dick Hamlet, *Member, IEEE,* and Ross Taylor

This study was undertaken because partition testing did not live up to its intuitive value in two earlier studies. In their brief for random testing [3], Duran and Ntafos published a precise comparison between it and partition testing. Their surprising result is that the two methods are of almost equal value, under assumptions that seem to favor partition testing. Random testing has a decidedly spotty reputation, probably because it makes almost no use of special information about the program being tested. It is certainly counterintuitive that the best systematic method is little improvement over the worst. Hamlet [5] corroborates this result using a different sampling model. He shows random testing to be *superior* to partition testing, its superiority *increasing* with more partitions and with the program confidence required.

"Whitebox Fuzzing"

~~Partition Testing~~ Does Not Inspire Confidence

Dick Hamlet, *Member, IEEE,* and Ross Taylor

This study was undertaken because partition testing did not live up to its intuitive value in two earlier studies. In their brief for random testing [3], Duran and Ntafos published a precise comparison between it and partition testing. Their surprising result is that the two methods are of almost equal value, under assumptions that seem to favor partition testing. Random testing has a decidedly spotty reputation, probably because it makes almost no use of special information about the program being tested. It is certainly counterintuitive that the best systematic method is little improvement over the worst. Hamlet [5] corroborates this result using a different sampling model. He shows random testing to be *superior* to partition testing, its superiority *increasing* with more partitions and with the program confidence required.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

If our whitebox fuzzer takes too long per input, our blackbox fuzzer outperforms!

» There is a maximum time per test input!

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.

[FSE’14] **On the Efficiency of Automated Testing,** M Böhme, S. Paul,
[TSE’15] **A Probabilistic Analysis of the Efficiency of Automated Testing,** M Böhme, S. Paul; IEEE Trans. Softw. Eng.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

If our whitebox fuzzer takes too long per input, our blackbox fuzzer outperforms!

» There is a maximum time per test input!

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  On 100 machines, it takes **63 milliseconds.**

[FSE'14] **On the Efficiency of Automated Testing,** M Böhme, S. Paul,
[TSE'15] **A Probabilistic Analysis of the Efficiency of Automated Testing,** M Böhme, S. Paul; IEEE Trans. Softw. Eng.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

So, if we have sufficiently many machines (to maximize execs/sec), blackbox fuzzers are the best we can get, right?

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.

  On my machine, this takes **6.3 seconds**.

  On 100 machines, it takes **63 milliseconds.**

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

So, if we have sufficiently many machines (to maximize execs/sec), blackbox fuzzers are the best we can get, right?

Wrong.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  On 100 machines, it takes **63 milliseconds.**

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

So, if we have sufficiently many machines (to maximize execs/sec), blackbox fuzzers are the best we can get, right?

Wrong.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.

Instead of generating inputs from scratch, can we reuse existing inputs?

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

So, if we have sufficiently many machines (to maximize execs/sec), blackbox fuzzers are the best we can get, right?

Wrong.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.

Instead of generating inputs from scratch, can we reuse existing inputs?

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

So, if we have sufficiently many machines (to maximize execs/sec), blackbox fuzzers are the best we can get, right?

Wrong.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

Alright, we cheated. We chose a good seed to start with.

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

Alright, we cheated. We chose a good seed to start with.

What if we could automatically discover this seed?

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- **Generational** Blackbox Fuzzer: Discovers the bug after **4 billion inputs**, in expectation.
- **Mutational** Blackbox Fuzzer mutates a random character in a seed.
  - Started with the seed **bad?**
  - Discovers the bug after $((4^{-1})*(2^{-8}))^{-1} \approx$ **1024 inputs**, in expectation.

↑

Greybox Fuzzing: “Enumerate”

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!

[CCS’16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

Greybox Fuzzing: “Enumerate”

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!

Seed corpus	“Interesting” Input	Expected #inputs
****	b***	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b***	ba**	$(1/2 \times 4^{-1} \times 2^{-8})^{-1}$ = 2048
**** b*** ba**	bad*	$(1/3 \times 4^{-1} \times 2^{-8})^{-1}$ = 3072
**** b*** ba** bad*	bad!	$(1/4 \times 4^{-1} \times 2^{-8})^{-1}$ = 4096
		Total: 10240

[CCS’16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

Greybox Fuzzing: “Enumerate”

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!

****	b***	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b***	ba**	$(1/2 \times 4^{-1} \times 2^{-8})^{-1}$ = 2048
**** b*** ba**	bad*	$(1/3 \times 4^{-1} \times 2^{-8})^{-1}$ = 3072
**** b*** ba** bad*	bad!	$(1/4 \times 4^{-1} \times 2^{-8})^{-1}$ = 4096
		Total: 10240

[CCS’16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

Greybox Fuzzing: “Enumerate”

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == ‘!’)
          crash = 1;

  if(crash == 1) abort();
}
```

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!
- Greybox Fuzzing started only with `****` in the seed corpus discovers the bug after 10k inputs (in 150 microseconds)!

`****`	`b***`	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
`****` `b***`	`ba**`	$(1/2 \times 4^{-1} \times 2^{-8})^{-1}$ = 2048
`****` `b***` `ba**`	`bad*`	$(1/3 \times 4^{-1} \times 2^{-8})^{-1}$ = 3072
`****` `b***` `ba**` `bad*`	`bad!`	$(1/4 \times 4^{-1} \times 2^{-8})^{-1}$ = 4096
		Total: 10240

[CCS’16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

Greybox Fuzzing: “Enumerate”

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!
- Greybox Fuzzing started only with **** in the seed corpus discovers the bug after 10k inputs (in 150 microseconds)!
- **Boosted** Greybox Fuzzing started with **** in the seed corpus discovers the bug after 4k inputs (in 55 microseconds)!

`****`	`b***`	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
`****` `b***`	`ba**`	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024

`****`
`b***`
`ba**`
	`bad*`	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024

`****`
`b***`
`ba**`
`bad*`
	`bad!`	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024

		Total: 4096

[CCS’16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

More Machines!

Awesome! We have a really efficient fuzzers.
Let's throw more machines at the problem!

- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  → On 100 machines, it takes **63 milliseconds.**

More Machines!

X times more machines means
X times more bugs, right?

- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  On 100 machines, it takes **63 milliseconds.**

More Machines!

X times more machines means
X times more bugs, right?

Wrong.

- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1}$ ≈ **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  On 100 machines, it takes **63 milliseconds.**

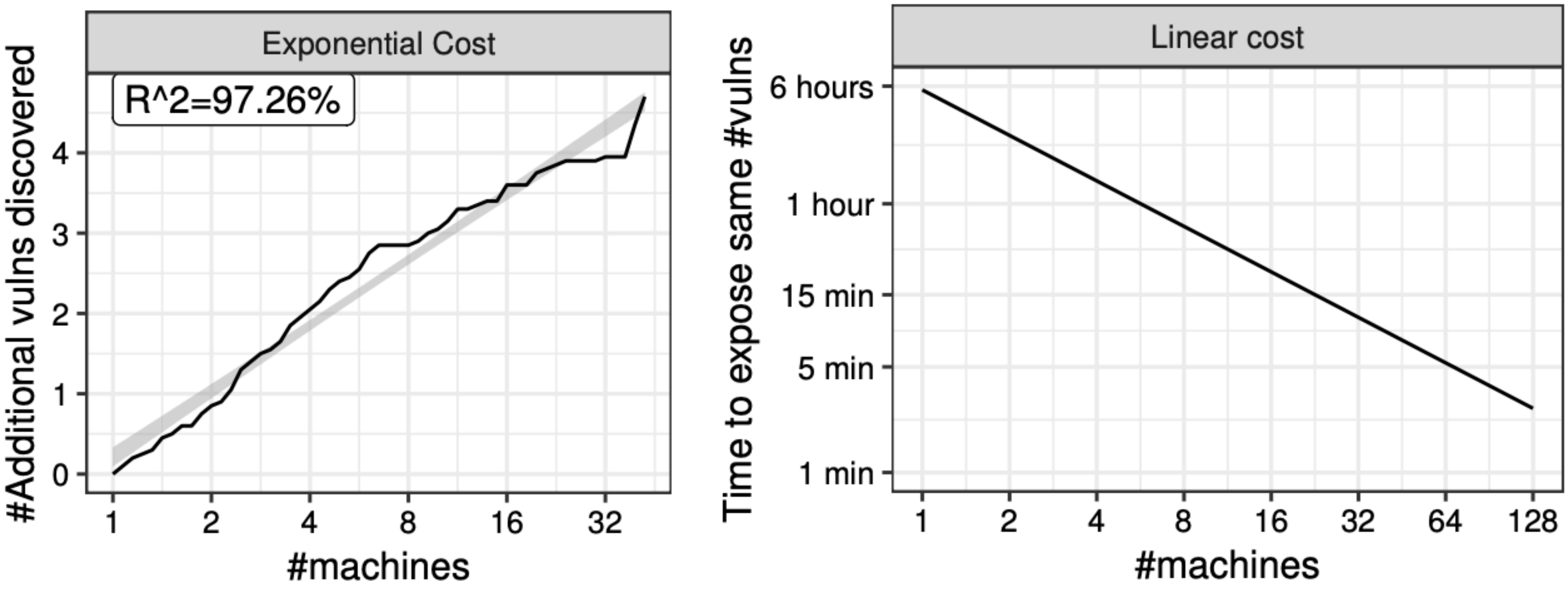


Figure 1: Each vuln. discovery requires exponentially more machines (left). Yet, exponentially more machines allow to find the same vulnerabilities exponentially faster (right).

Fuzzer Test Suite (45min campaigns)

Explaining Exponential Cost

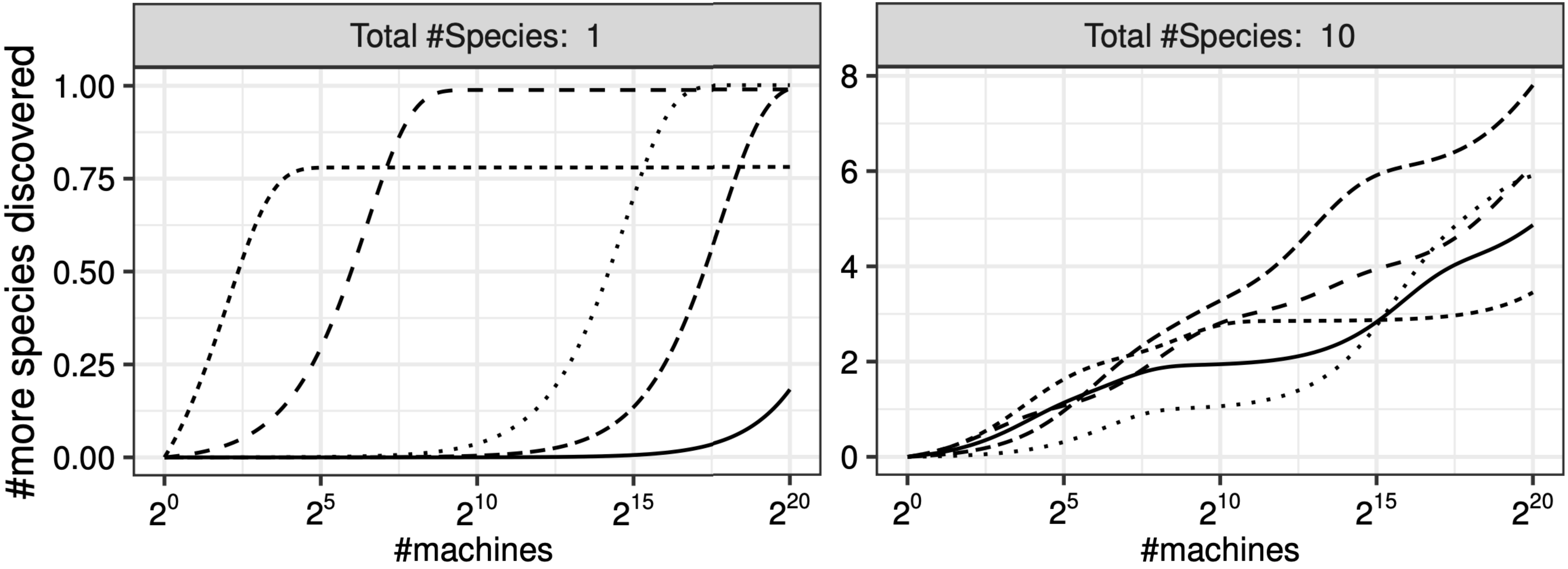


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Explaining Exponential Cost

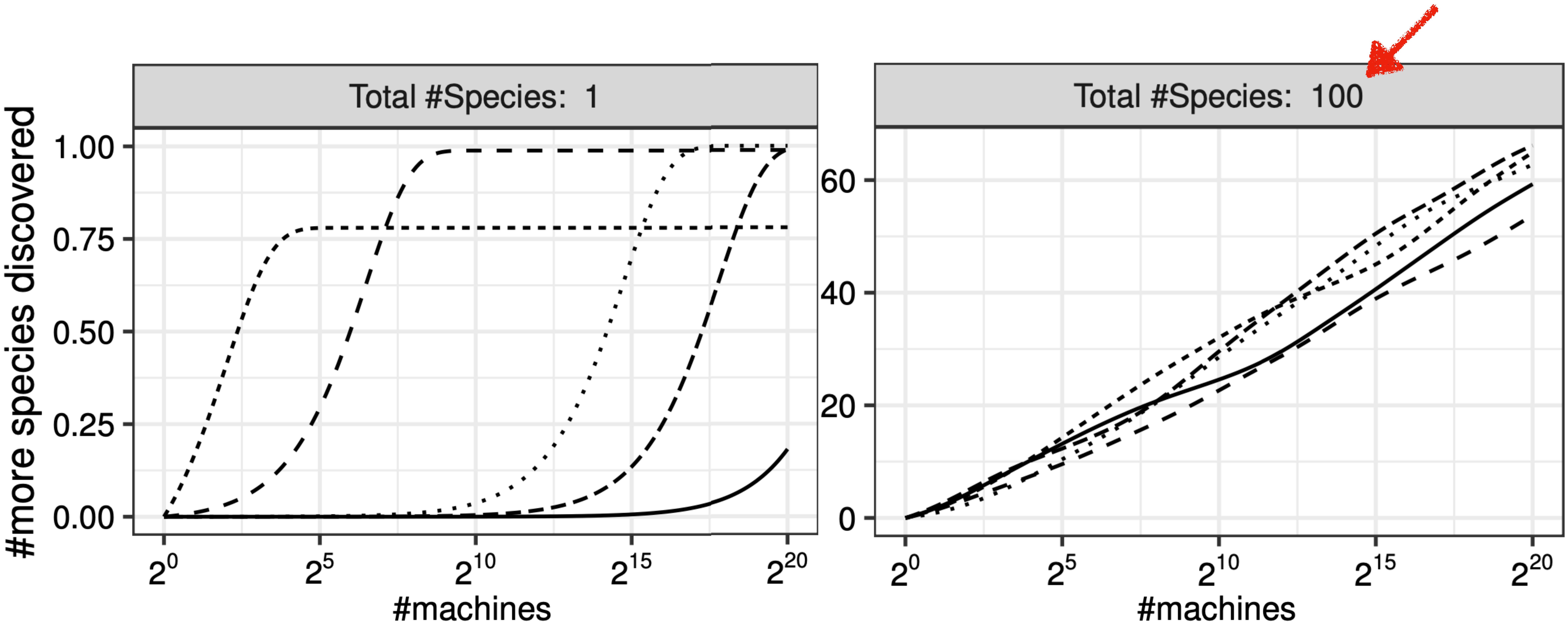


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Explaining Exponential Cost

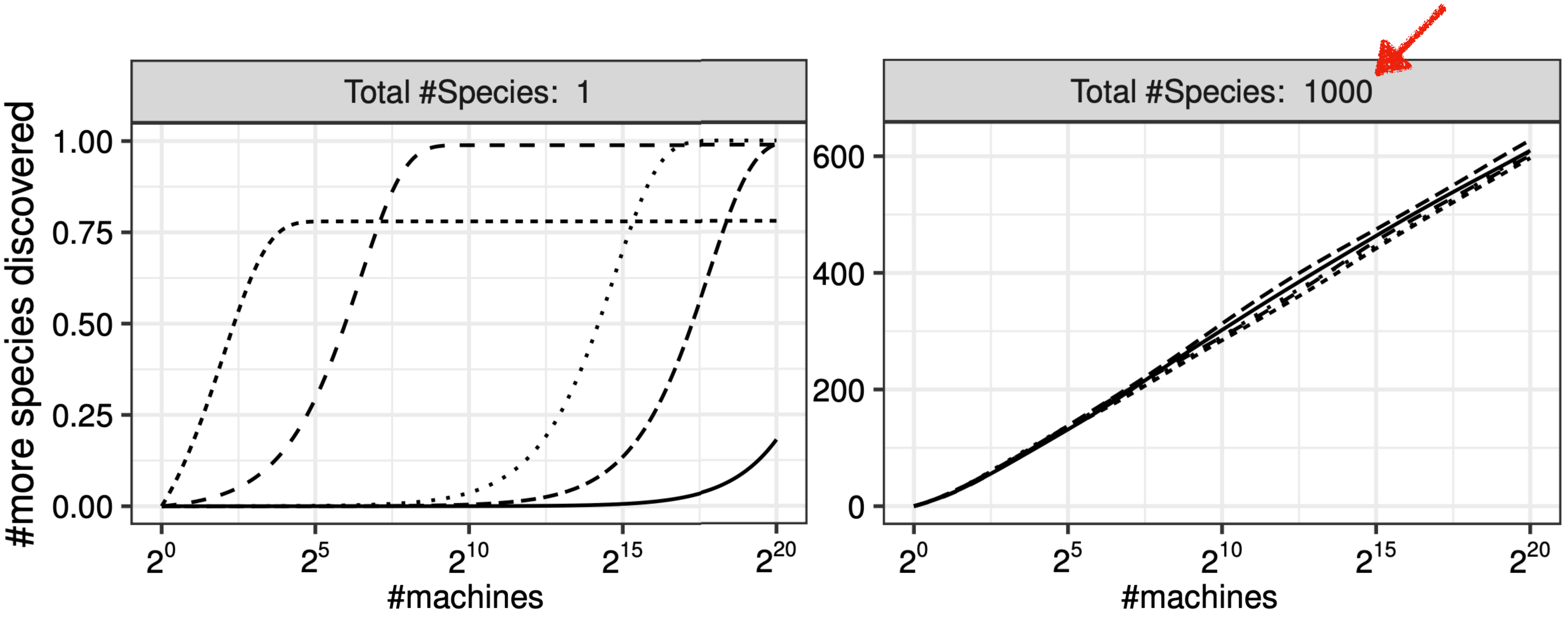


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Explaining Exponential Cost

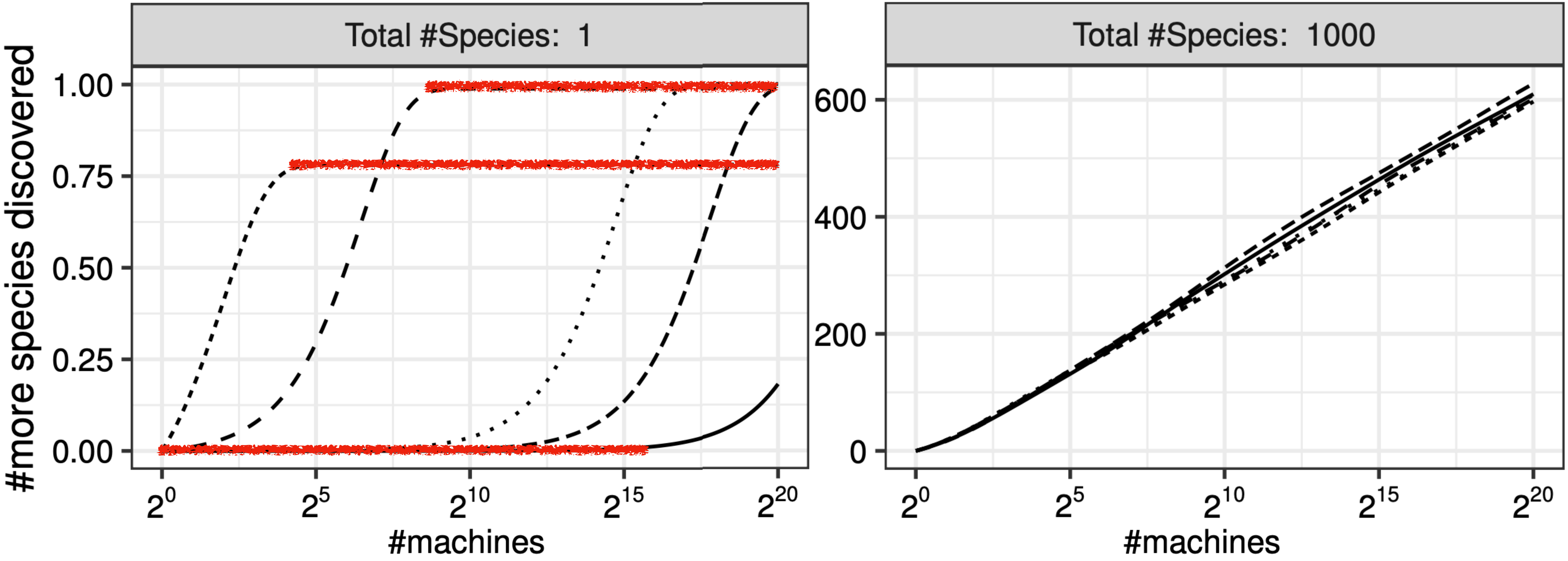


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Explaining to Exponential Cost

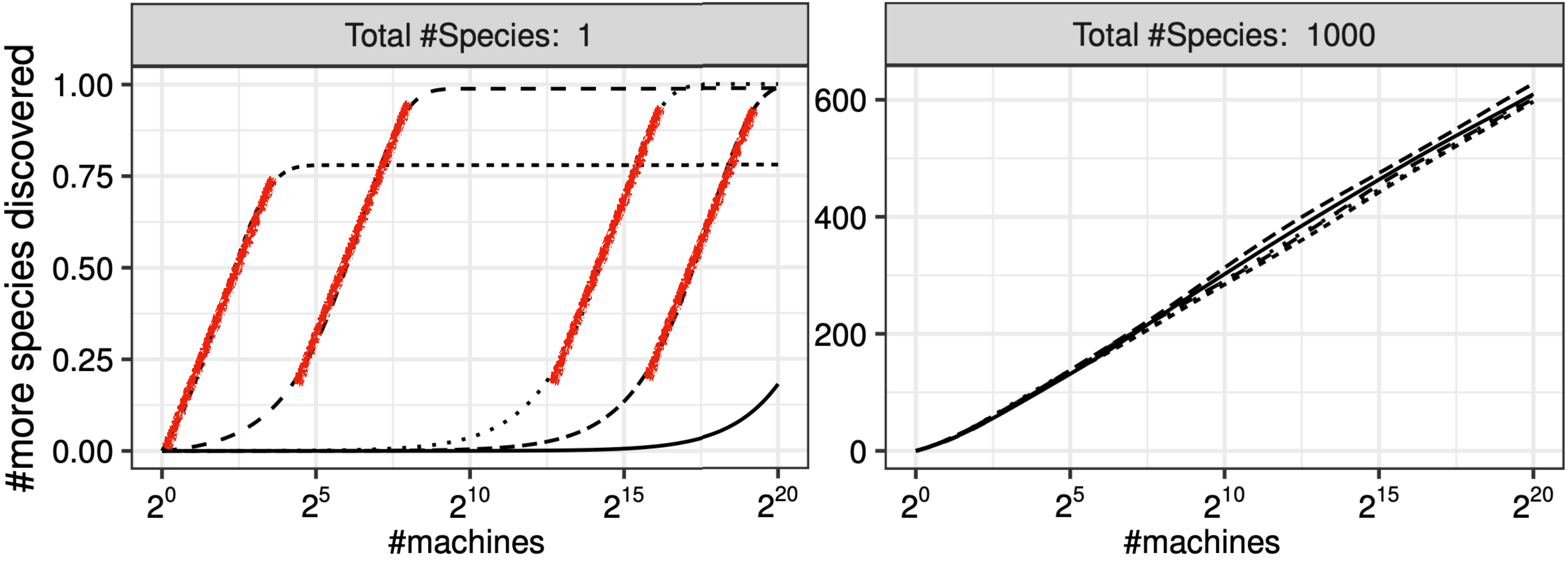
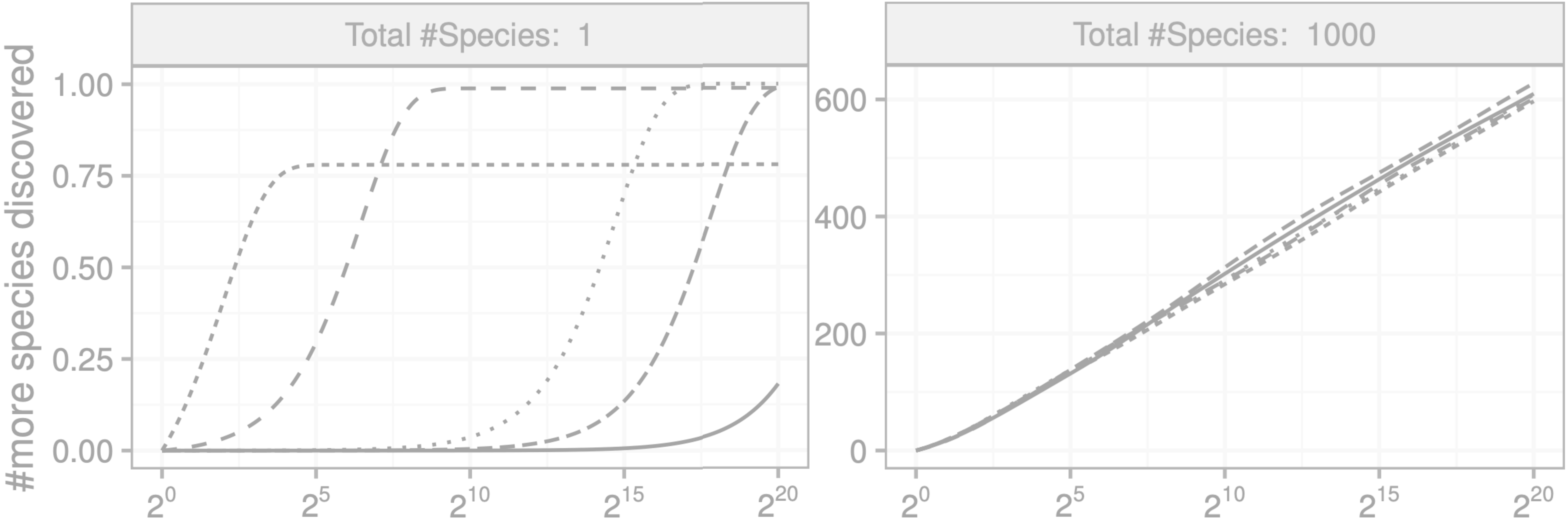


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Explaining to Exponential Cost



→ Intuitively, each new vulnerability requires some more resources (time or machines) than the previous vulnerability.

On the Cost of Vulnerability Discovery

A constant rate of vulnerability discovery requires exponential amount of resources.

*This is a fundamental limitation of fuzzing!

[FSE'20] **Fuzzing: On the Exponential Cost of Vulnerability Discovery.**
M. Böhme, Brandon Falk (Microsoft)
Nominated for ACM Distinguished Paper Award

Whitebox Fuzzing: Most Effective!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

2

← It can prove the absence of assertion violation, by enumerating all paths and modulo some assumptions.

Path Conditions

- ✓ φ_1 = (s0 != 'b')
- ✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')
- ✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')
- ✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 != '!')
- ✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd') /\ (s3 == '!')

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

If our whitebox fuzzer takes too long per input, our blackbox fuzzer outperforms!

» There is a maximum time per test input!

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.
  On my machine, this takes **6.3 seconds**.
  On 100 machines, it takes **63 milliseconds.**

Greybox Fuzzing: "Enumerate"

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!
- Greybox Fuzzing started only with **** in the seed corpus discovers the bug after 10k inputs (in 150 microseconds)!
- **Boosted** Greybox Fuzzing started with **** in the seed corpus discovers the bug after 4k inputs (in 55 microseconds)!

****	b***	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b***	ba**	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b*** ba**	bad*	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b*** ba** bad*	bad!	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
		Total: 4096

[CCS'16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

Exponential Cost of Vulnerability Discovery

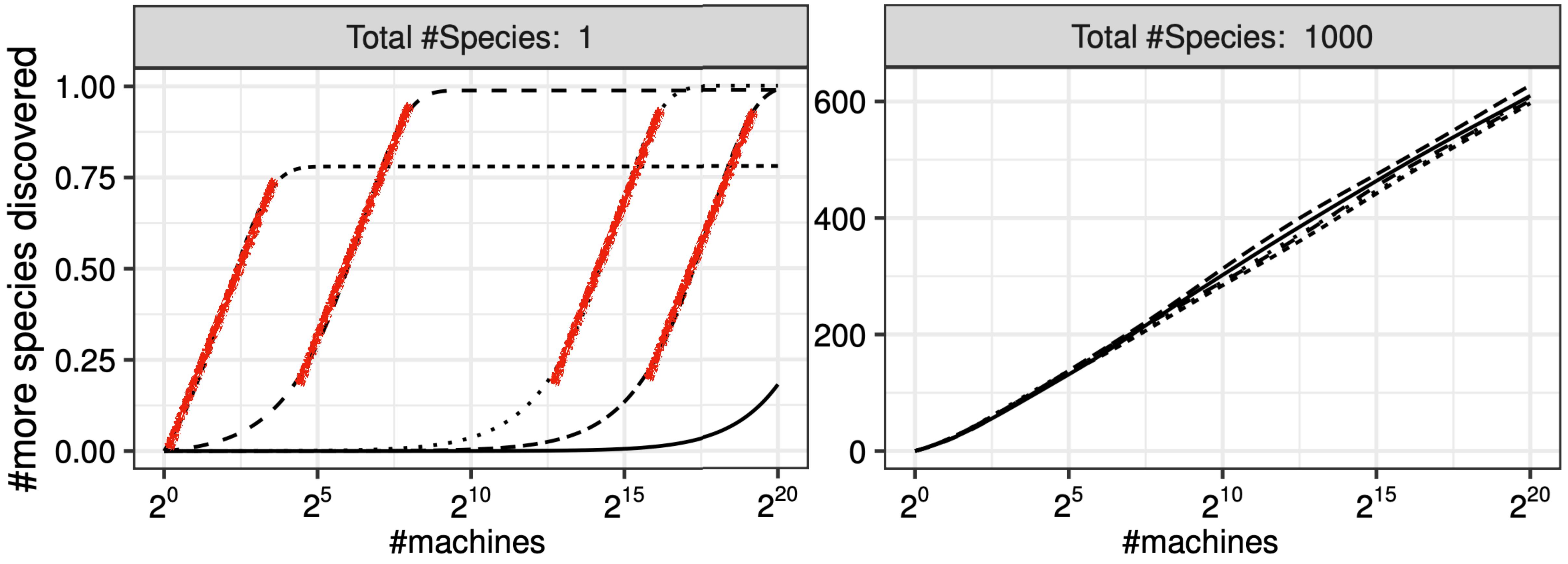


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.

Whitebox Fuzzing: Most Effective!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

2 ← It can prove the absence of assertion violation, by enumerating all paths and modulo some assumptions.

Path Conditions

✓ φ_1 = (s0 != 'b')
✓ φ_2 = (s0 == 'b') /\ (s1 != 'a')
✓ φ_3 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 != 'd')
✓ φ_4 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd')
✗ φ_5 = (s0 == 'b') /\ (s1 == 'a') /\ (s2 == 'd')

Blackbox Fuzzing: Super fast!

```
void crashme (char s0, char s1, char s2, char s3) {
  int crash = 0;

  if (s0 == 'b')
    if (s1 == 'a')
      if (s2 == 'd')
        if (s3 == '!')
          crash = 1;

  if(crash == 1) abort();
}
```

If our whitebox fuzzer takes too long per input, our blackbox fuzzer outperforms!

» There is a maximum time per test input!

- Whitebox Fuzzer: Discovers the bug after **3 inputs**, in expectation.
- Blackbox Fuzzer: Discovers the bug after $((1/256)^4)^{-1} \approx$ **4 billion inputs**, in expectation.
- ...akes **6.3 seconds.**
- ...akes **63 milliseconds.**

If you want to take a deeper dive:

- Read our interactive text book: **The Fuzzing Book**
- Read our IEEE Software article: **“Fuzzing: Challenges and Reflections”**
- Apply for PhD / PostDoc in **my group at MPI-SP**, Bochum, Germany.

Web: https://mboehme.github.com Twitter: @mboehme_

Greybox Fuzzing: “Enu...

- **Greybox Fuzzing**: Add generated inputs to the corpus which increase coverage!
- Greybox Fuzzing started only with **** in the seed corpus discovers the bug after 10k inputs (in 150 microseconds)!
- **Boosted** Greybox Fuzzing started with **** in the seed corpus discovers the bug after 4k inputs (in 55 microseconds)!

****	b***	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b***	ba**	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b*** ba**	bad*	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
**** b*** ba** bad*	bad!	$(1 \times 4^{-1} \times 2^{-8})^{-1}$ = 1024
		Total: 4096

[CCS'16] **Coverage-based Greybox Fuzzing as Markov Chain**
M Böhme, V.T. Pham, A. Roychoudhury (extended in IEEE TSE journal)

...Vulnerability Discovery

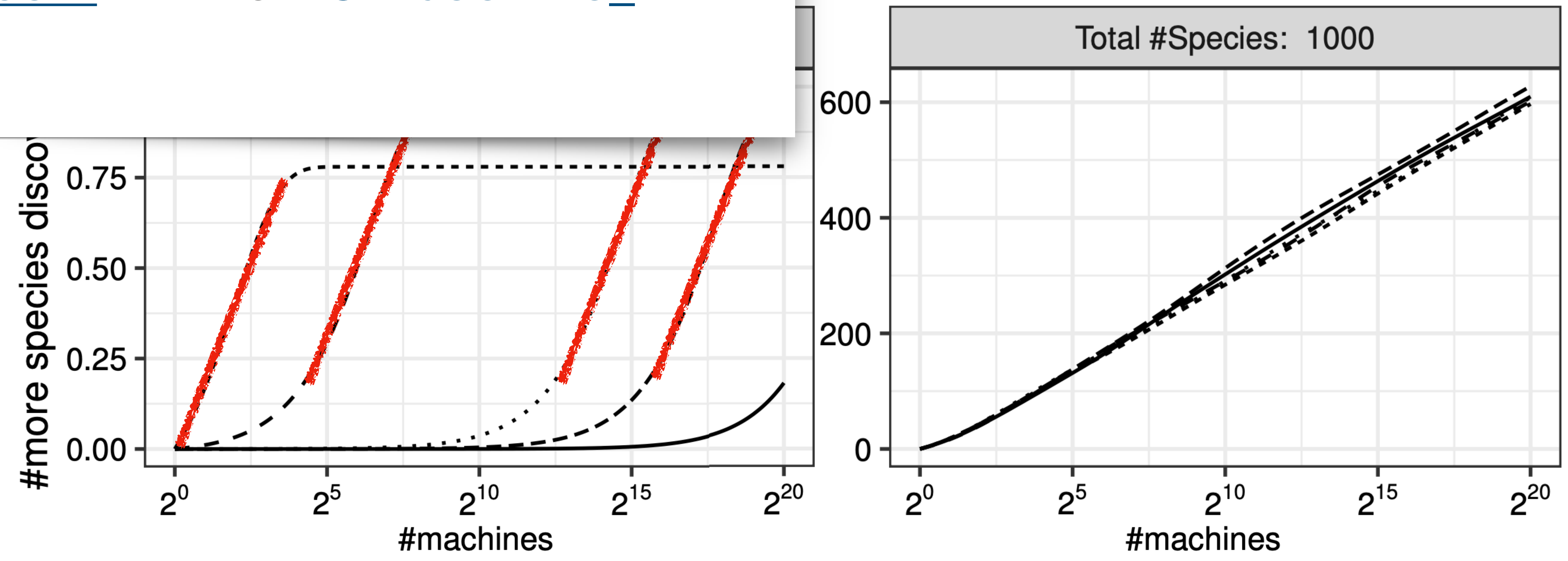


Figure 10. Number of additional species discovered in a fixed time budget as the number of machines increases (5 random samples of $\{q_i\}_{i=1}^{S}$ each.