

*L3 Mention Informatique
Parcours Informatique et MIAGE*

Génie Logiciel Avancé - Advanced Software Engineering

White-Box Fuzz-Tests

Burkhart Wolff
wolff@lri.fr

Billions and Billions of Constraints: Whitebox Fuzz Testing in Production

Ella Bounimova, Patrice Godefroid, **David Molnar**
Microsoft Research

Microsoft Fixes 21 Security Bugs in Windows, IE, Office

Each bug like this costs Microsoft ~USD 1 million

If you're unlucky, it could cost you too...

Many such bugs are "corner cases" in C/C++ code

File parsers: video, audio, pictures...

```
int foo(int x) { // x is an input
    int y = x + 3;
    if (y == 13) abort(); // error
    return 0;
}
```

```
int foo(int x) { // x is an input
    int y = x + 3;
    if (y == 13) abort(); // error
    return 0;
}
```

Random choice of x: one chance in 2^{32} to find error
“Fuzz testing” Widely used, remarkably effective!

```
int foo(int x) { // x is an input
    int y = x + 3;
    if (y == 13) abort(); // error
    return 0;
}
```

Core idea:

- 1) Pick an arbitrary “seed” input
- 2) Record path taken by program executing on “seed”
- 3) Create symbolic abstraction of path and generate tests

```
int foo(int x) { // x is an input
    int y = x + 3;
    if (y == 13) abort(); // error
    return 0;
}
```

Example:

- 1) Pick x to be 5
- 2) Record $y = 5 + 3 = 8$, record program tests " $8 \neq 13$ "
- 3) Symbolic *path condition*: " $x + 3 \neq 13$ "

```
void top(char input[4])
```

```
{
```

```
    int cnt = 0;
```

```
    if (input[0] == 'b') cnt++;
```

```
    if (input[1] == 'a') cnt++;
```

```
    if (input[2] == 'd') cnt++;
```

```
    if (input[3] == '!') cnt++;
```

```
    if (cnt >= 3) crash();
```

```
}
```

input = "good"

Path constraint:

$I_0 \neq 'b' \rightarrow I_0 = 'b'$

$I_1 \neq 'a' \rightarrow I_1 = 'a'$

$I_2 \neq 'd' \rightarrow I_2 = 'd'$

$I_3 \neq '!' \rightarrow I_3 = '!'$



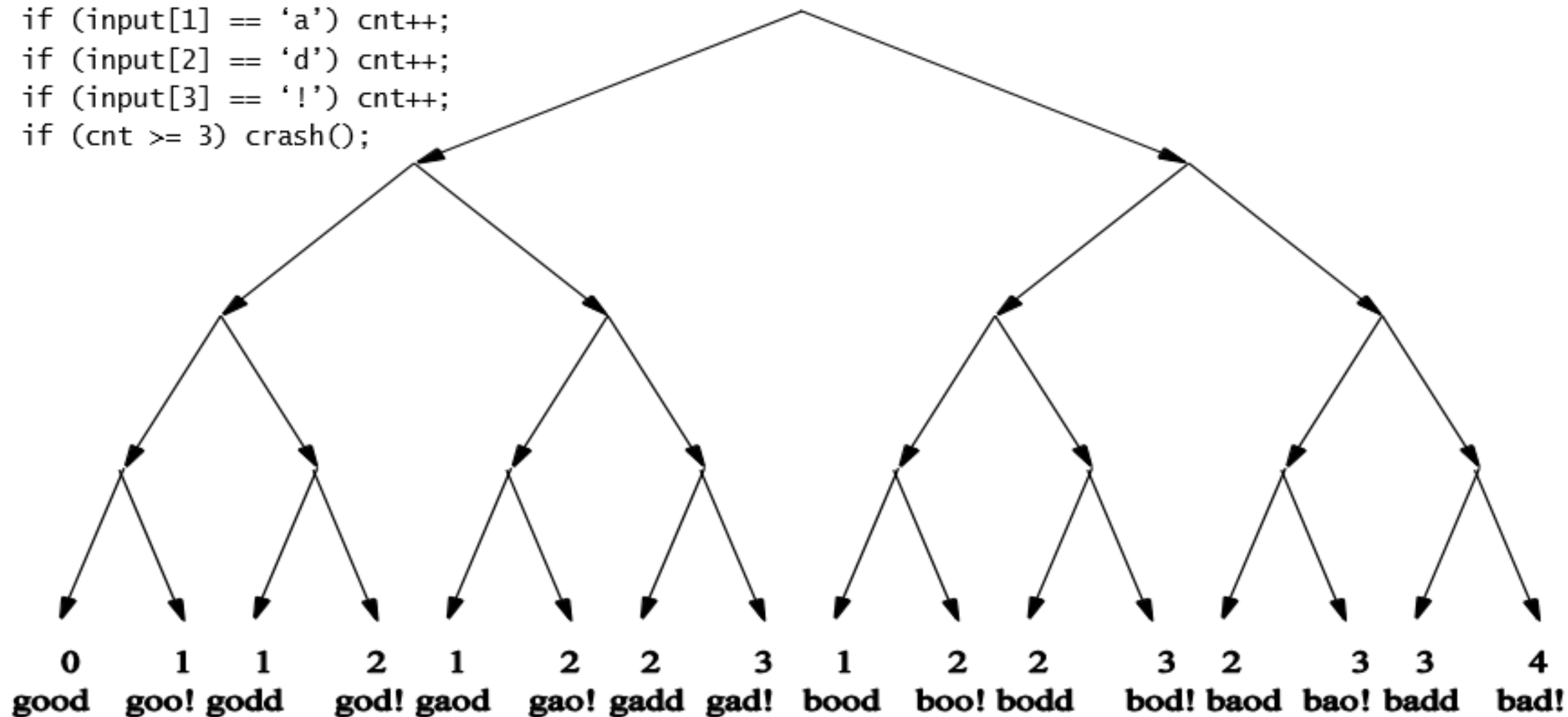
Negate each constraint in path constraint

Solve new constraint \rightarrow new input


```

void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}

```



```
11: mov eax, inp1
    mov cl,  inp2
    shl eax,  cl
    jnz 12
    jmp          13
12:  div          ebx,  eax
//  Is  this  safe  ?
//  Is  eax  !=  0  ?
13:  ...
```

Work with x86 **binary code** on Windows
Leverage full-instruction-trace recording

Pros:

- If you can run it, you can analyze it
- Don't care about build processes
- Don't care if source code available

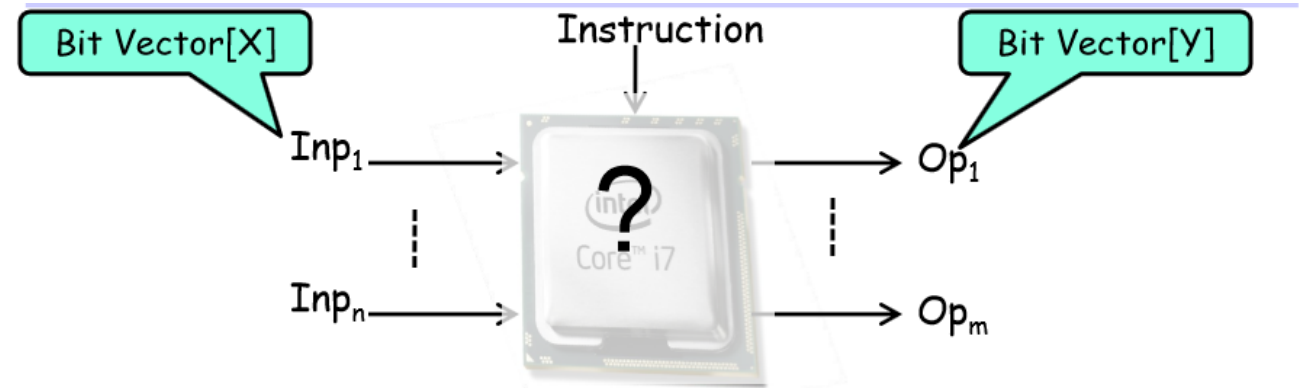
Cons:

- Lose programmer's intent (e.g. types)
- Hard to "see" string manipulation, memory object graph manipulation, etc.

SHLD—Double Precision Shift Left (Continued)

Operation

```
COUNT ← COUNT MOD 32;
SIZE ← OperandSize
IF COUNT = 0
  THEN
    no operation
  ELSE
    IF COUNT ≥ SIZE
      THEN (* Bad parameters *)
        DEST is undefined;
        CF, OF, SF, ZF, AF, PF are undefined;
      ELSE (* Perform the shift *)
        CF ← BIT[DEST, SIZE - COUNT];
        (* Last bit shifted out on exit *)
        FOR i ← SIZE - 1 DOWNTO COUNT
          DO
            Bit(DEST, i) ← Bit(DEST, i - COUNT);
          OD;
        FOR i ← COUNT - 1 DOWNTO 0
          DO
            BIT[DEST, i] ← BIT[Src, i - COUNT + SIZE];
          OD;
        FI;
      FI;
    FI;
```



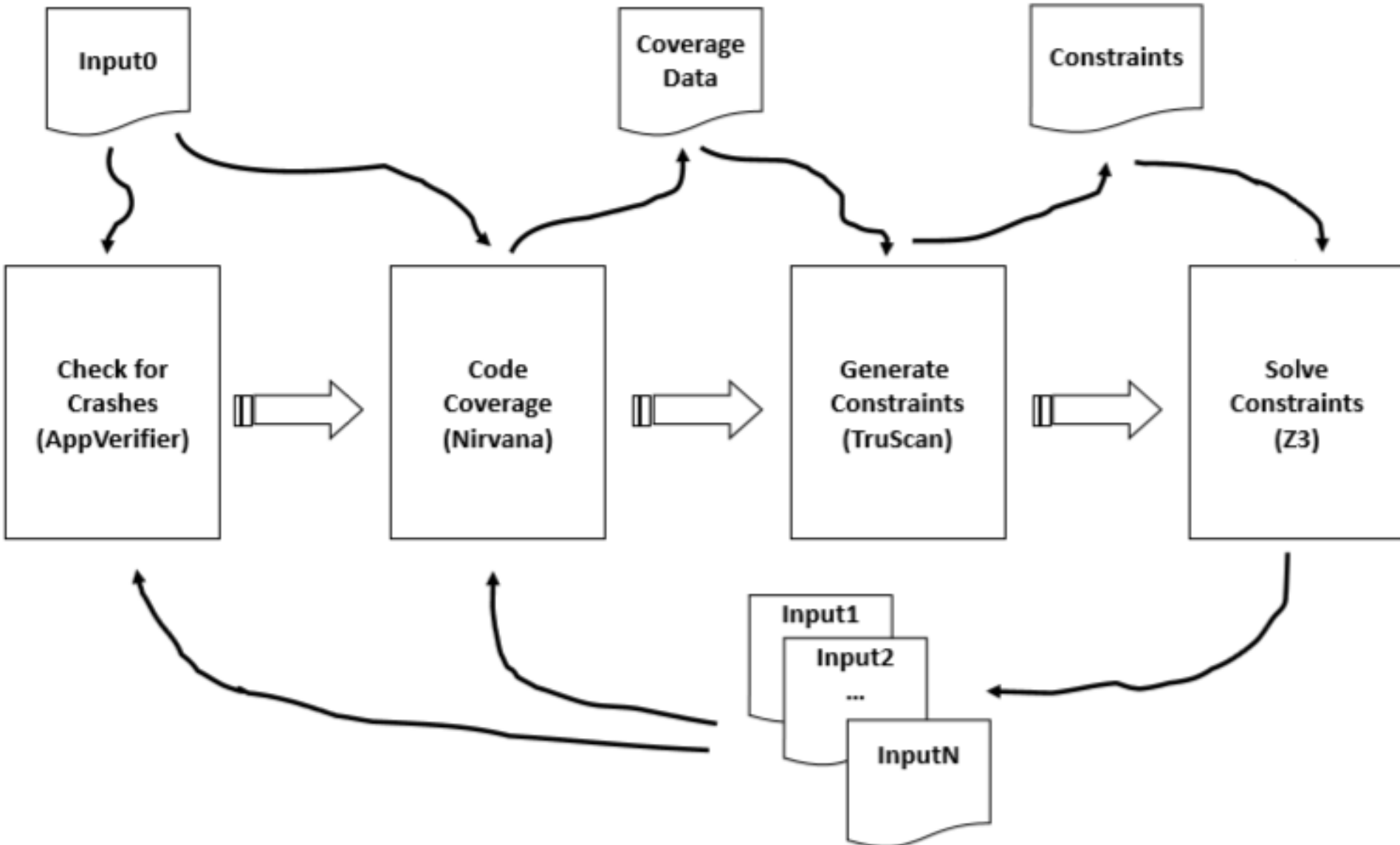
Hand-written models (so far)

Uses Z3 support for non-linear operations

Normally “concretize” memory accesses where address is symbolic

# instructions executed	1,455,506,956
# instr. executed after 1st read from file	928,718,575
# constraints generated (full path constraint)	25,958
# constraints dropped due to cache hits	244,170
# constraints dropped due to limit exceeded	193,953
# constraints satisfiable (= # new tests)	2,980
# constraints unsatisfiable	22,978
# constraint solver timeouts (>5 secs)	0
symbolic execution time (secs)	2,745
constraint solving time (secs)	953

Figure 10: Statistics for the execution of the test suite on the test cases.

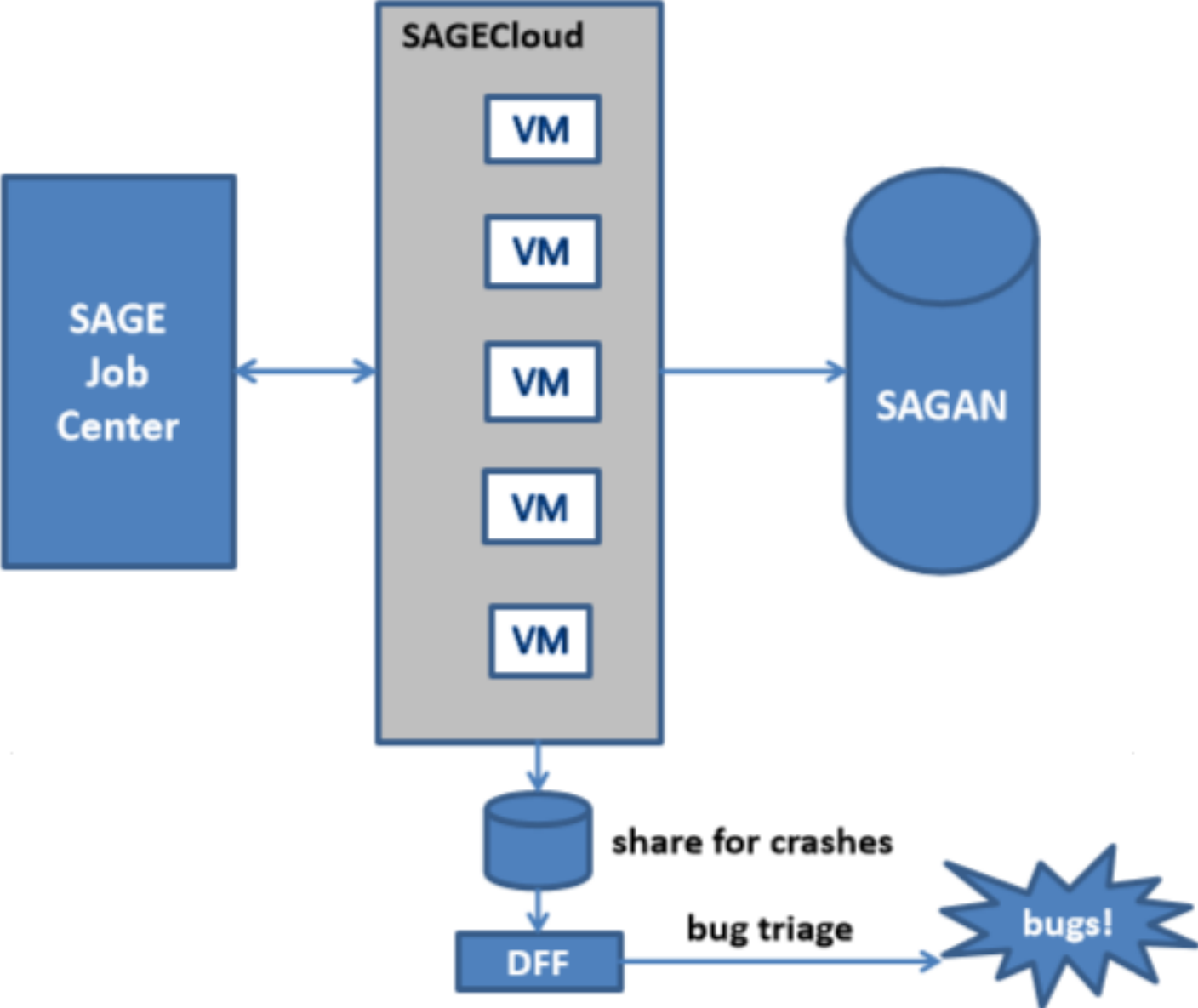


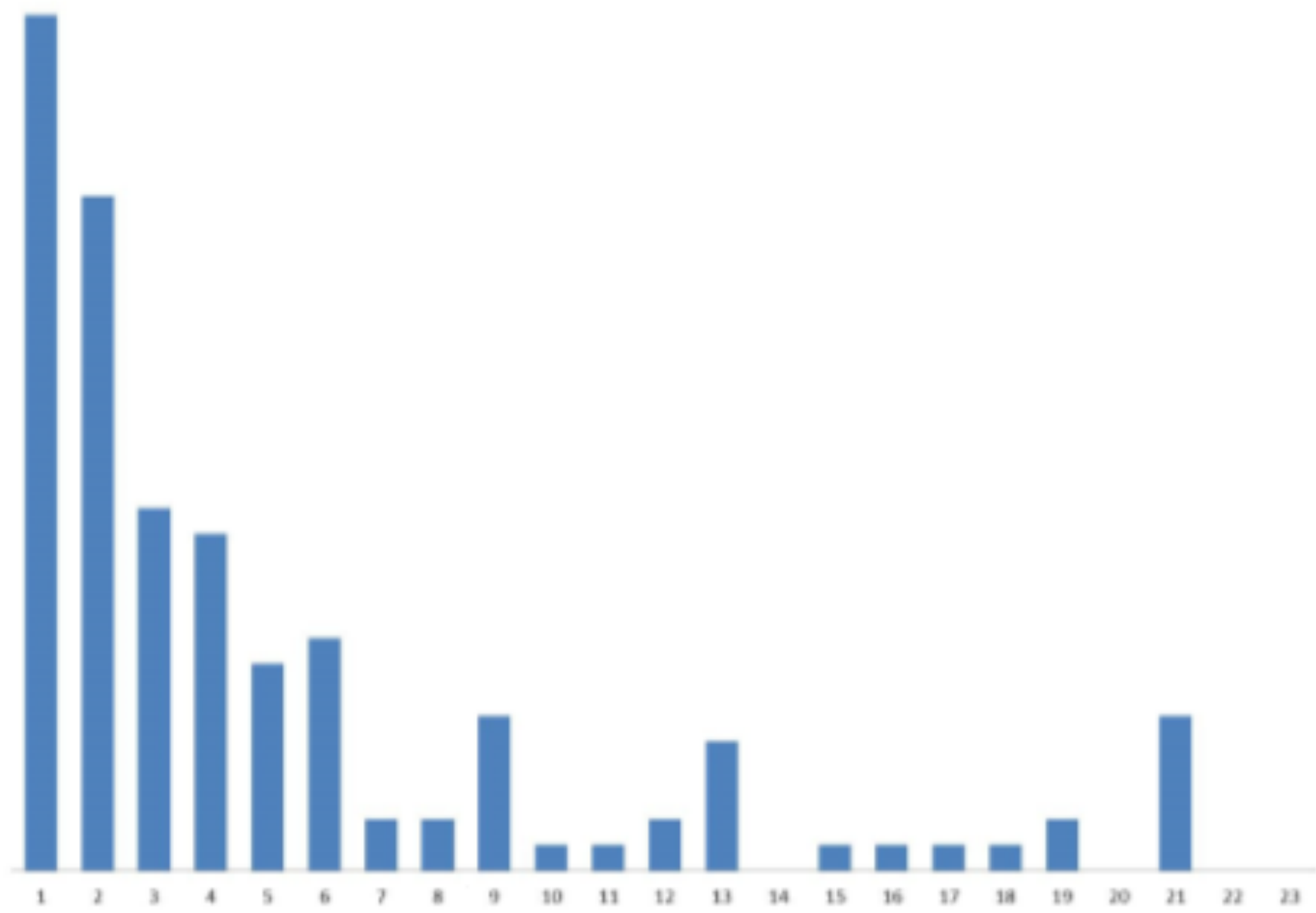
```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000060h: 00 00 00 00 ; ....
```

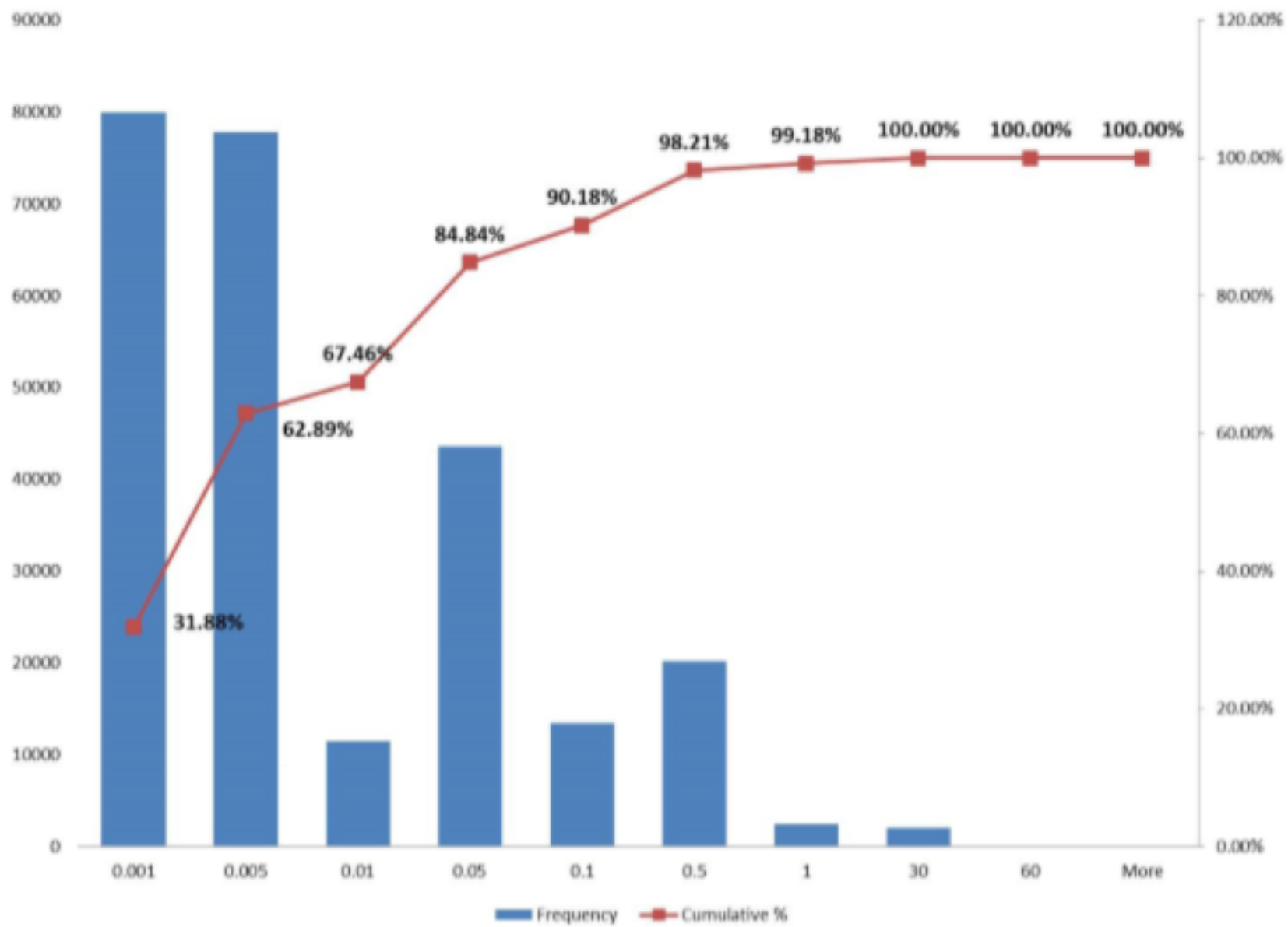
Generation 0 – seed file

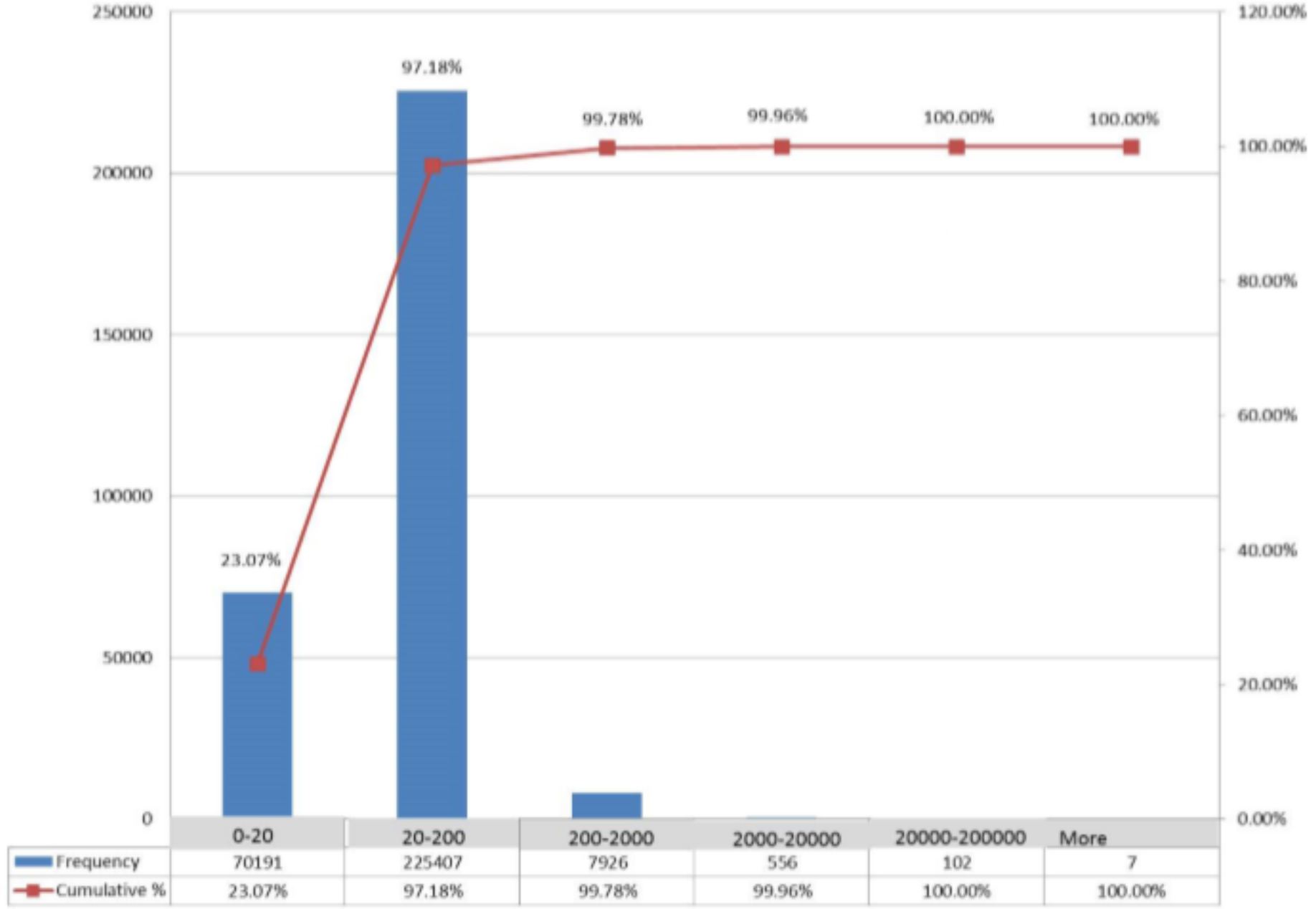
```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .....
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ...strh...vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ...strf2uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; .....
00000060h: 00 00 00 00 ; .....
```

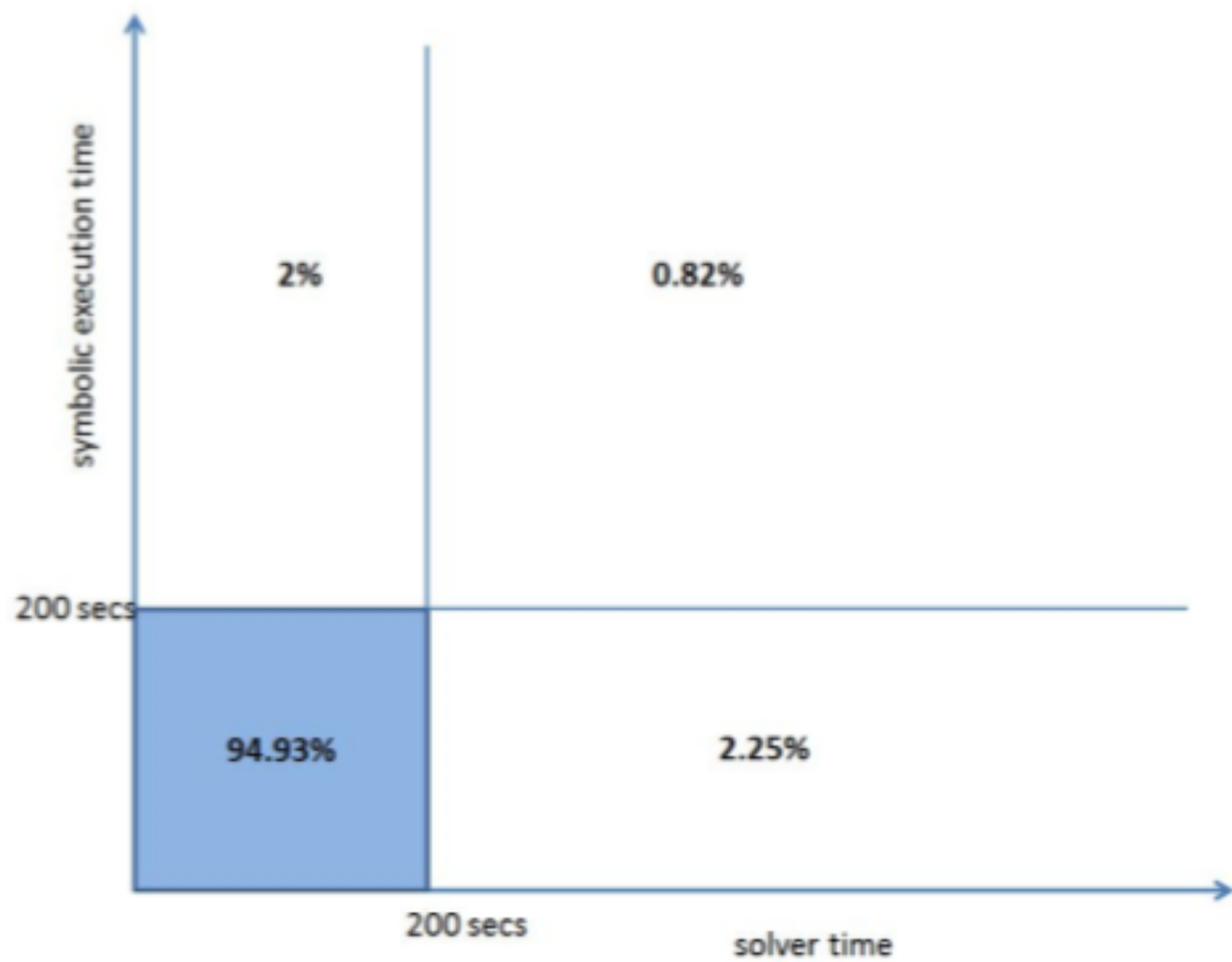
Generation 10 – crash bucket 1212954973!

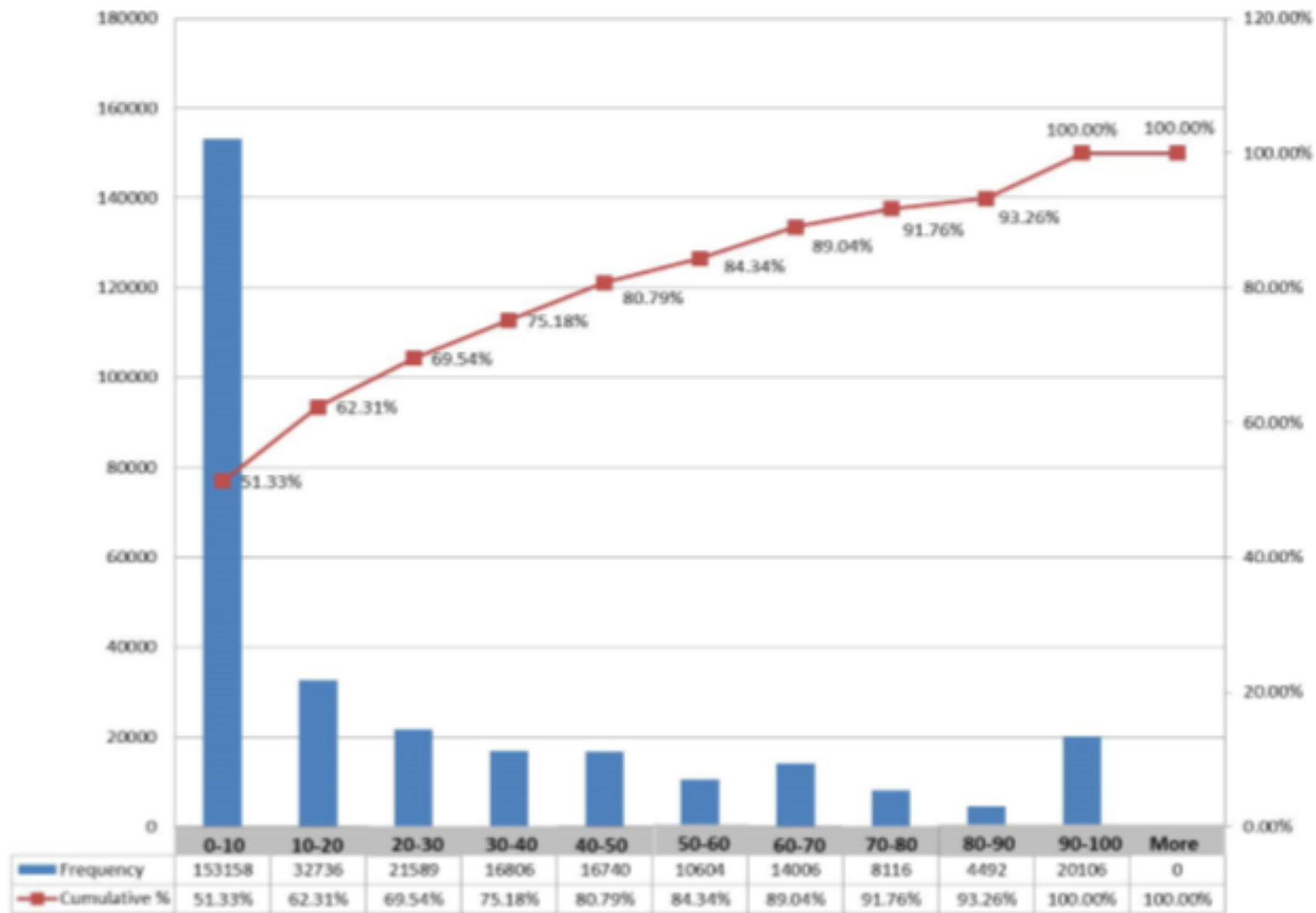


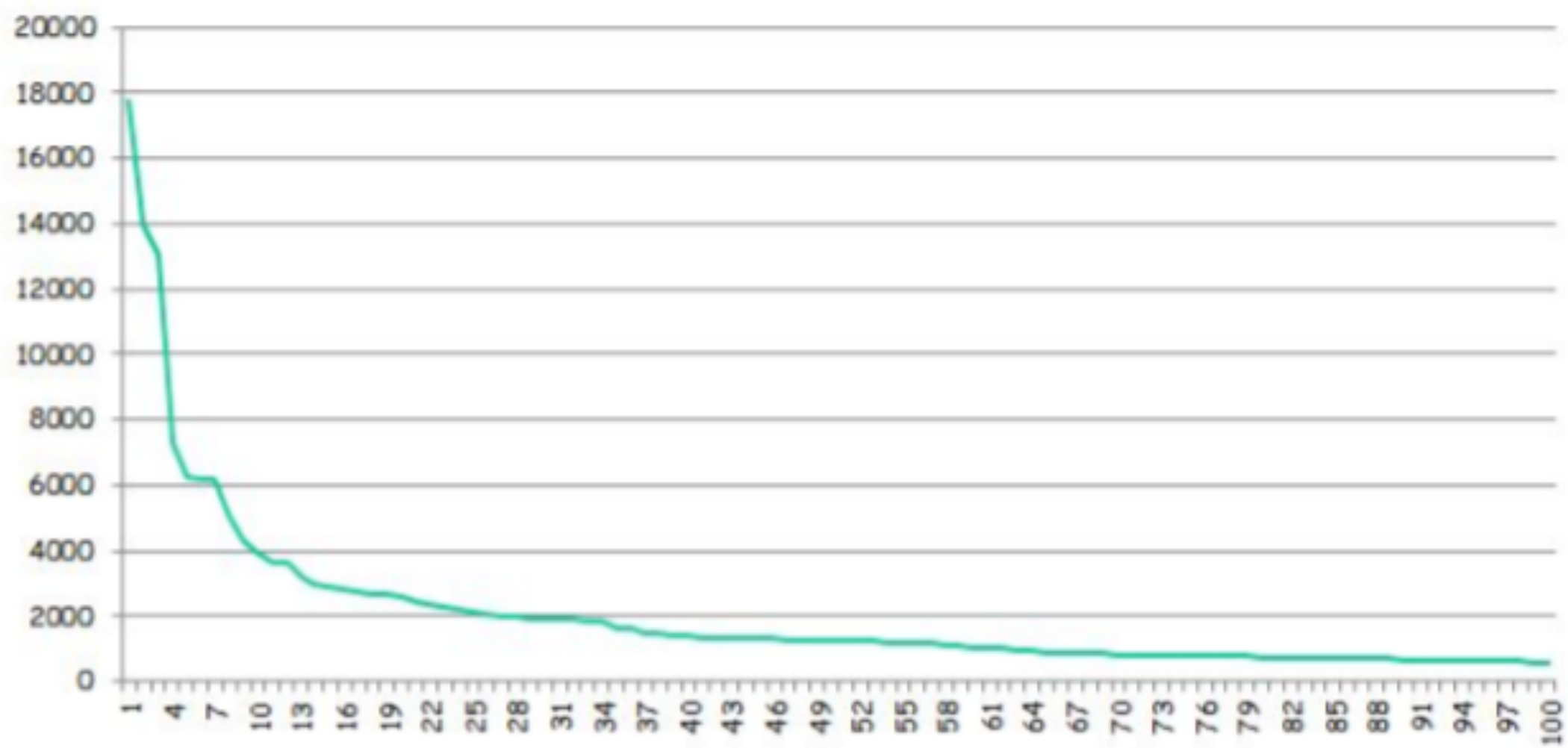












Reflections

Data invaluable for driving investment priorities

Can't cover all x86 instructions by hand – look at which ones are used!

Recent: synthesizing circuits from templates (Godefroid & Taly PLDI 2012)

Plus finds configuration errors, compiler changes, etc. impossible otherwise

Data can reveal test programs have special structure

Scaling too long traces needs careful attention to representation

Sometimes run out of memory on 4 GB machine with large programs

Even incomplete, unsound analysis useful because whole-program

SAGE finds bugs missed by all other methods

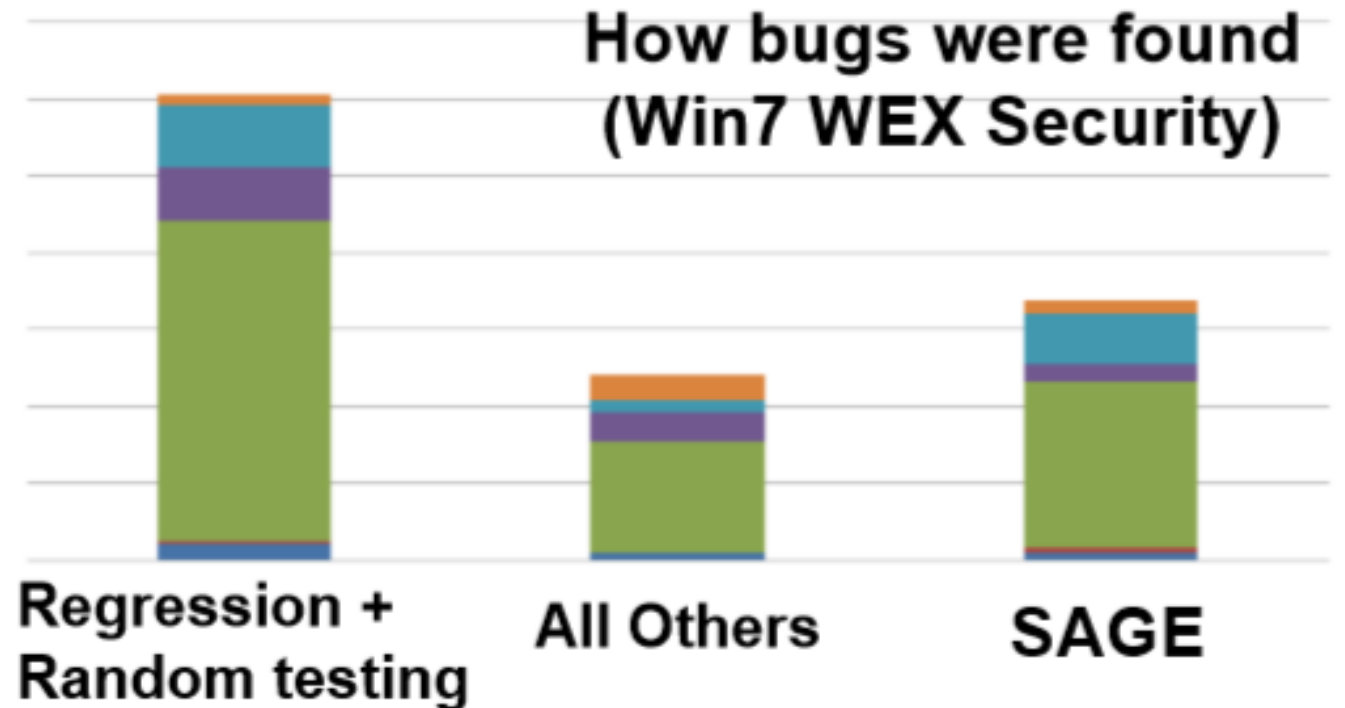
Supporting users & partners super important, a lot of work!

Payoff

3.4 billion constraints queried June 2010 – November 2012

Millions of test cases generated

Run daily on Office, Windows



Thank you!

Questions? dmolnar@microsoft.com