

# EXCVATE: Spoofing Exceptions and Solving Constraints to Test Exception Handling in Numerical Libraries

**Jackson Vanover<sup>\*</sup>, James Demmel<sup>†</sup>, Xiaoye Sherry Li<sup>‡</sup>, Cindy Rubio-González<sup>\*</sup>**

<sup>\*</sup>University of California, Davis

<sup>†</sup>University of California, Berkeley

<sup>‡</sup>Lawrence Berkeley National Laboratory



U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under award DE-SC0020286,  
and the National Science Foundation under award CCF-1750983.

Numerical code should be correct

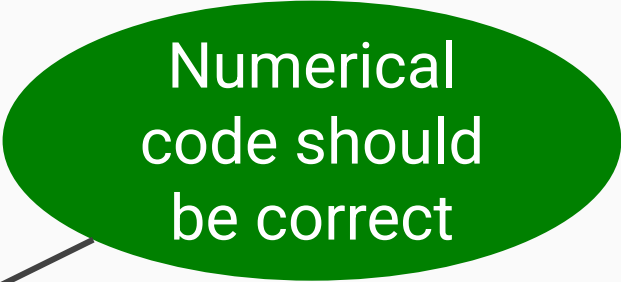
Numerical code should be correct  
but floating-point arithmetic is inherently error-prone.

Numerical code should be correct  
but floating-point arithmetic is inherently error-prone.

**OVERFLOW  
DIVIDE-BY-ZERO  
INVALID  
INEXACT  
UNDERFLOW**

Numerical  
code should  
be correct

Numerical  
code should  
be correct



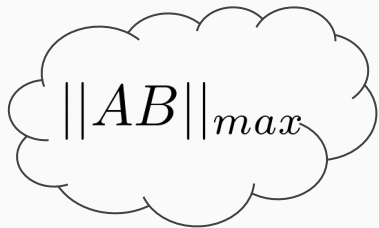
View exceptions as bugs:  
Make your code exception-free!

```
graph TD; A([Numerical code should be correct]) --> B[View exceptions as bugs: Make your code exception-free!]; A --> C[View exceptions as inevitabilities: Ensure correct exception handling!];
```

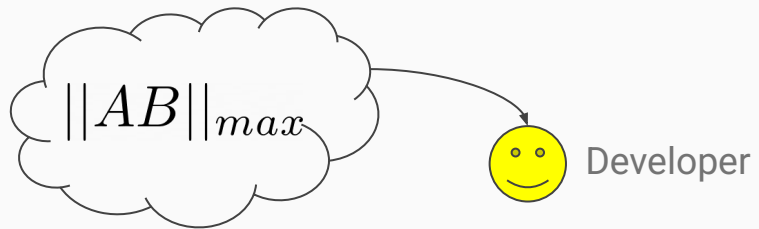
Numerical  
code should  
be correct

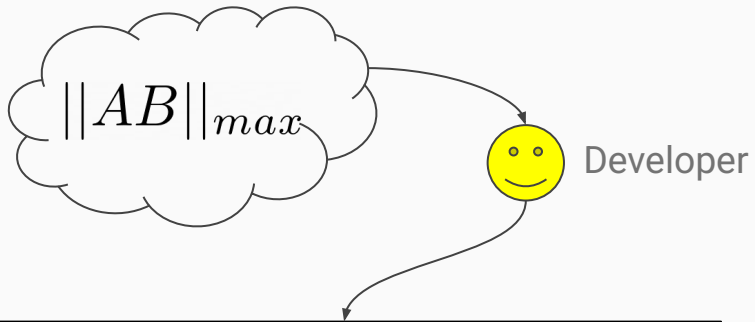
View exceptions as bugs:  
Make your code exception-free!

View exceptions as inevitabilities:  
Ensure correct exception handling!

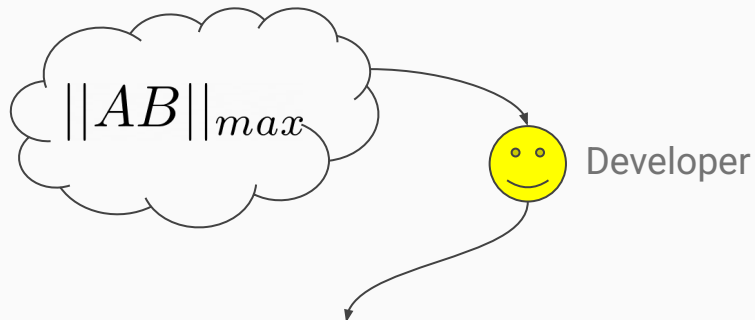

$$\|AB\|_{\max}$$





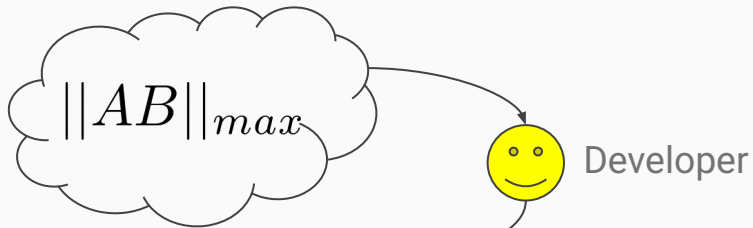


```
function mmMaxNorm(dim, A, B)
    ...
    result = 0.0
    do i = 1, dim
        do j = 1, dim
            acc = 0.0
            do k = 1, dim
                acc = acc + A(i,k) * B(k,j)
            end do
            result = max(result, abs(acc))
        end do
    end do
    ...
end function mmMaxNorm
```



```
function mmMaxNorm(dim, A, B)
    ...
    result = 0.0
    do i = 1, dim
        do j = 1, dim
            acc = 0.0
            do k = 1, dim
                acc = acc + A(i,k) * B(k,j)
            end do
            result = max(result, abs(acc))
        end do
    end do
    ...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

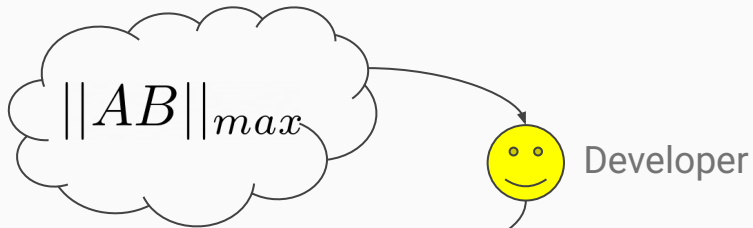


```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1



```
function mmMaxNorm(dim, A, B)
```

```
...
```

```
result = 0.0
```

```
do i = 1, dim
```

```
  do j = 1, dim
```

```
    acc = 0.0
```

```
    do k = 1, dim
```

```
      acc = acc + A(i,k) * B(k,j)
```

```
    end do
```

```
    result = max(result, abs(acc))
```

```
  end do
```

```
end do
```

```
...
```

```
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

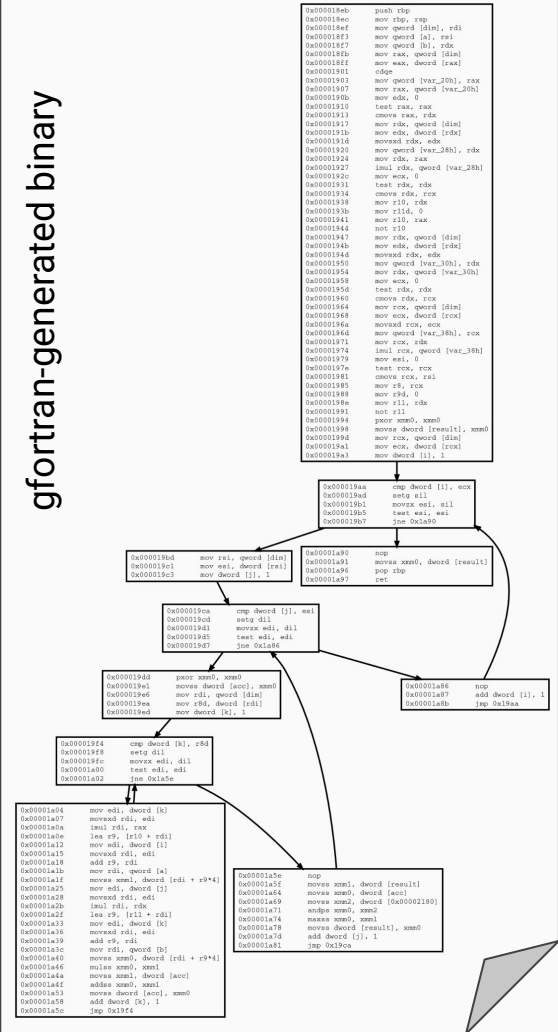
gfortran

bin1

0.0

**INCORRECT**

# gfortran-generated binary

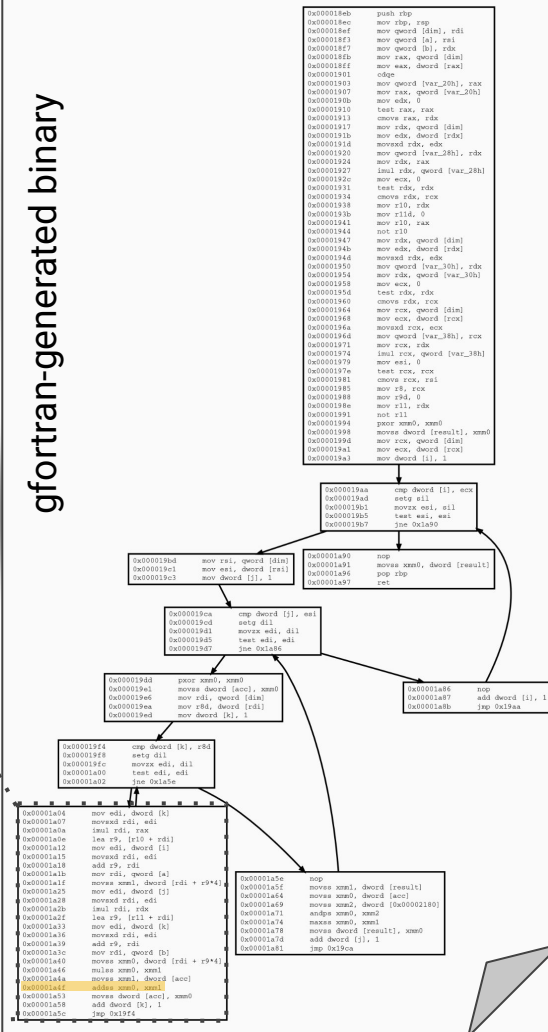


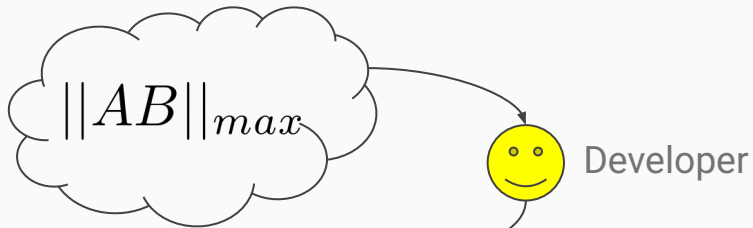
The eighth execution of this add instruction triggers an “Invalid” exception, generating a NaN that is ultimately lost.

```

0x00001a04    mov edi, dword [k]
0x00001a07    movsxd rdi, edi
0x00001a0a    imul rdi, rax
0x00001a0e    lea r9, [r10 + rdi]
0x00001a12    mov edi, dword [i]
0x00001a15    movsxd rdi, edi
0x00001a18    add r9, rdi
0x00001a1b    mov rdi, qword [a]
0x00001a1f    movss xmm1, dword [rdi + r9*4]
0x00001a25    mov edi, dword [j]
0x00001a28    movsxd rdi, edi
0x00001a2b    imul rdi, rdx
0x00001a2f    lea r9, [r11 + rdi]
0x00001a33    mov edi, dword [k]
0x00001a36    movsxd rdi, edi
0x00001a39    add r9, rdi
0x00001a3c    mov rdi, qword [b]
0x00001a40    movss xmm0, dword [rdi + r9*4]
0x00001a46    mulss xmm0, xmm1
0x00001a4a    movss xmm1, dword [acc]
0x00001a4f    addss xmm0, xmm1
0x00001a53    movss dword [acc], xmm0
0x00001a58    add dword [k], 1
0x00001a5c    jmp 0x19f4
  
```

gfortran-generated binary





```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

gfortran

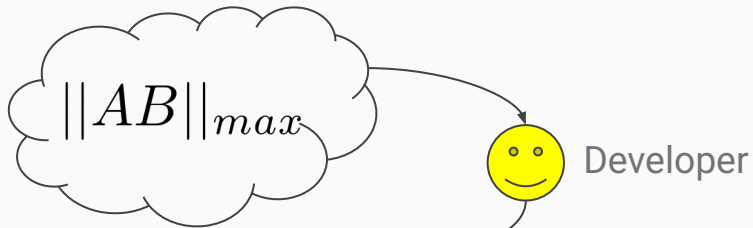
bin1

0.0

**INCORRECT**

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$
$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$





```
function mmMaxNorm(dim, A, B)
```

```
...
```

```
result = 0.0
```

```
do i = 1, dim
```

```
  do j = 1, dim
```

```
    acc = 0.0
```

```
    do k = 1, dim
```

```
      acc = acc + A(i,k) * B(k,j)
```

```
    end do
```

```
    result = max(result, abs(acc))
```

```
  end do
```

```
end do
```

```
...
```

```
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

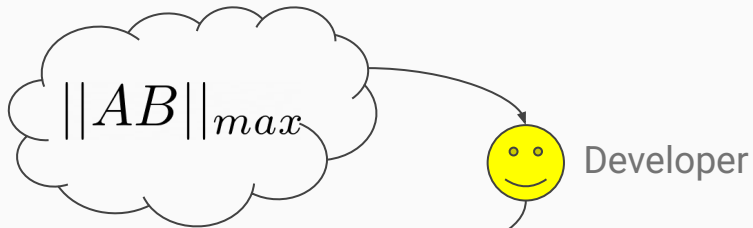
bin1

ifx

bin2

0.0

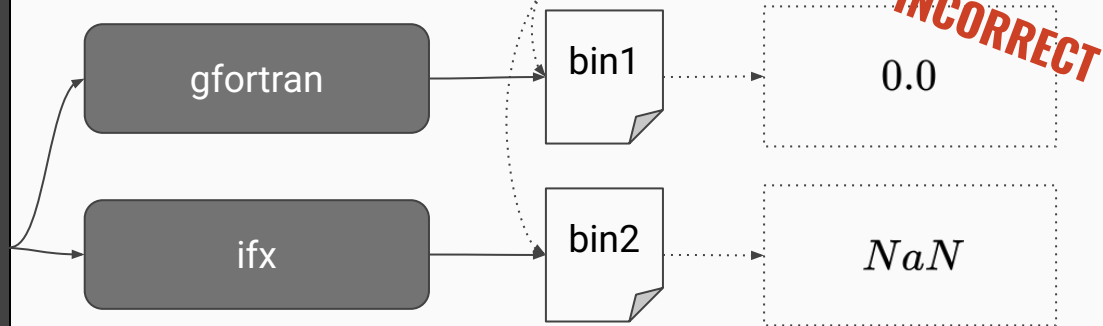
**INCORRECT**

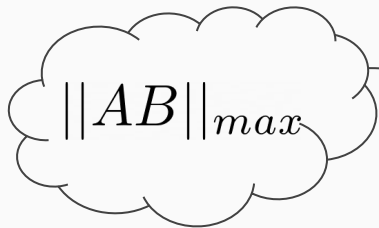


```
function mmMaxNorm(dim, A, B)
...
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
...
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$





Developer

```
function mmMaxNorm(dim, A, B)
```

```
...
```

```
result = 0.0
```

```
do i = 1, dim
```

```
  do j = 1, dim
```

```
    acc = 0.0
```

```
    do k = 1, dim
```

```
      acc = acc + A(i,k) * B(k,j)
```

```
    end do
```

```
    result = max(result, abs(acc))
```

```
  end do
```

```
end do
```

```
...
```

```
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1

0.0

**INCORRECT**

ifx

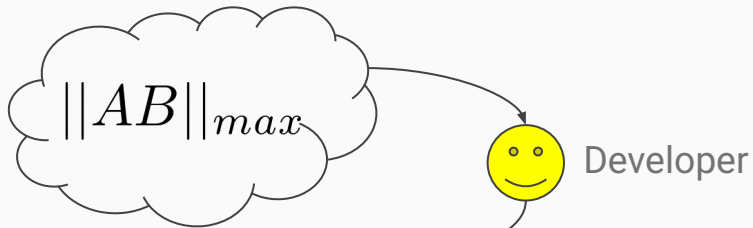
bin2

NaN

gfortran  
-fdefault-real-8

bin3

(increase FP precision to 64-bit)



```
function mmMaxNorm(dim, A, B)
```

```
...
```

```
result = 0.0
```

```
do i = 1, dim
```

```
  do j = 1, dim
```

```
    acc = 0.0
```

```
    do k = 1, dim
```

```
      acc = acc + A(i,k) * B(k,j)
```

```
    end do
```

```
    result = max(result, abs(acc))
```

```
  end do
```

```
end do
```

```
...
```

```
end function mmMaxNorm
```

$$A = \begin{bmatrix} 0.0 & 0.0 \\ -3.40282 \times 10^{38} & -2.4245 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0 & 3.06254 \times 10^{38} \\ 0.0 & -3.08694 \times 10^{38} \end{bmatrix}$$

gfortran

bin1

0.0

ifx

bin2

*NaN*

gfortran  
-fdefault-real-8

bin3

$1.04212 \times 10^{77}$

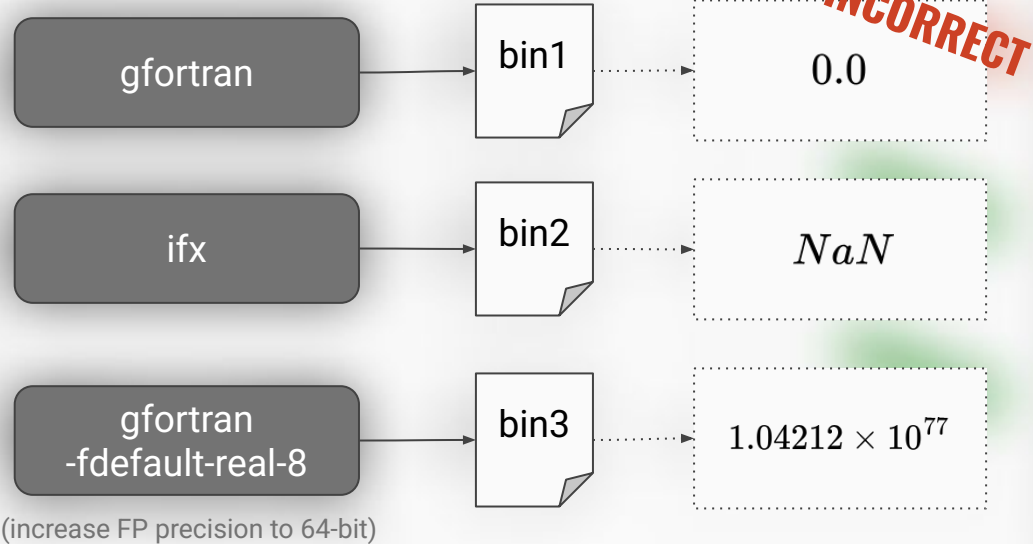
(increase FP precision to 64-bit)

**INCORRECT**

Numerical  
code should  
be correct

View exceptions as bugs:  
Make your code exception-free!

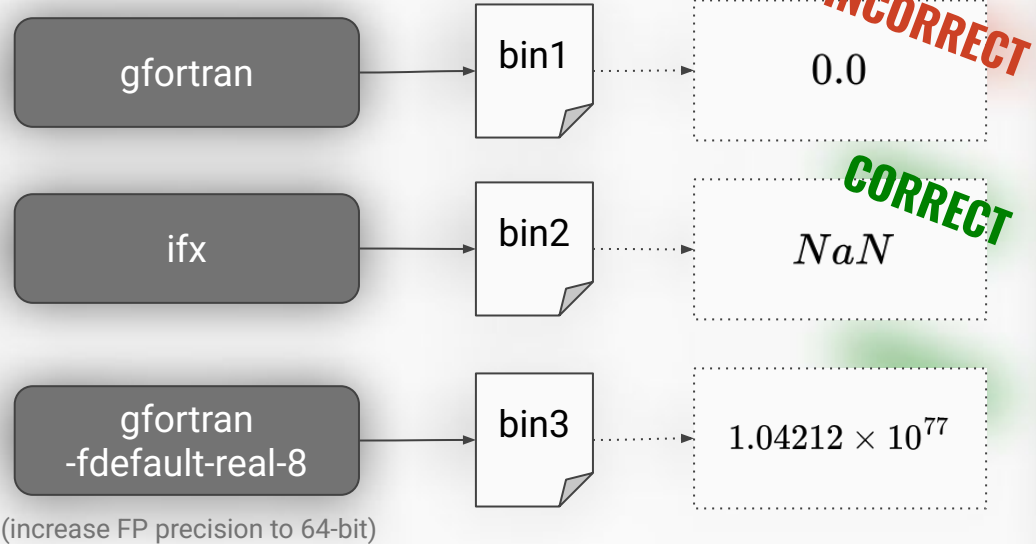
View exceptions as inevitabilities:  
Ensure correct exception handling!



Numerical  
code should  
be correct

View exceptions as bugs:  
Make your code exception-free!

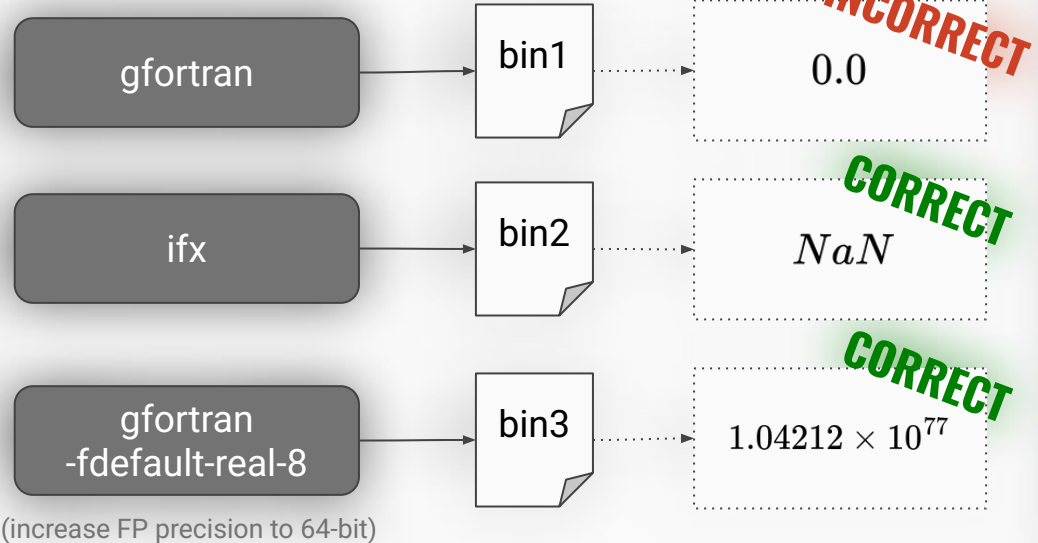
View exceptions as inevitabilities:  
Ensure correct exception handling!



Numerical  
code should  
be correct

View exceptions as bugs:  
Make your code exception-free!

View exceptions as inevitabilities:  
Ensure correct exception handling!



When an Overflow, Divide-by-Zero, or Invalid exception occurs, a **sound exception-handling policy** notifies users by either...

1. The presence of EVs in the output
2. Some library-specific mechanism triggered by checking for their presence at some point during the execution.

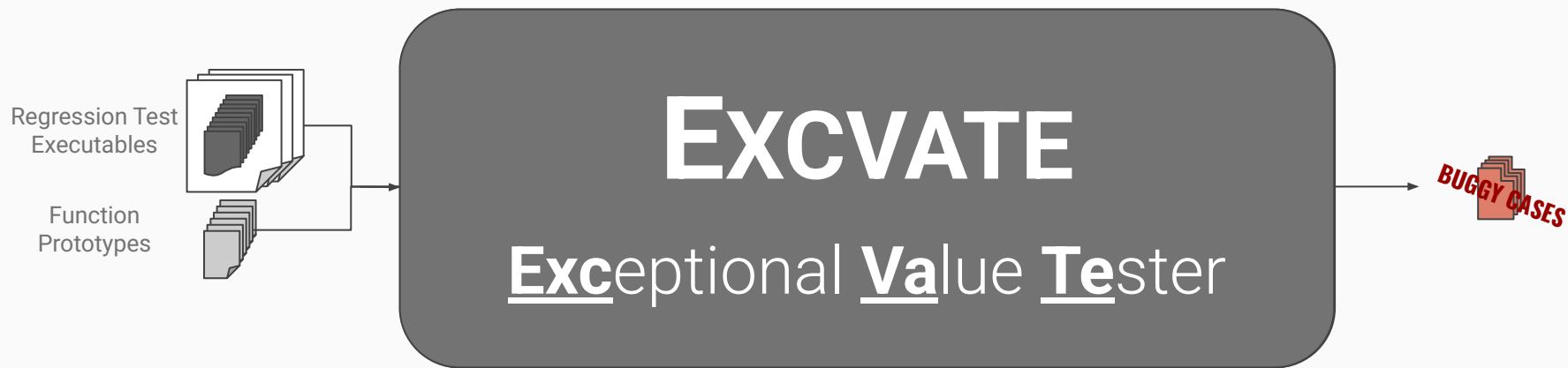


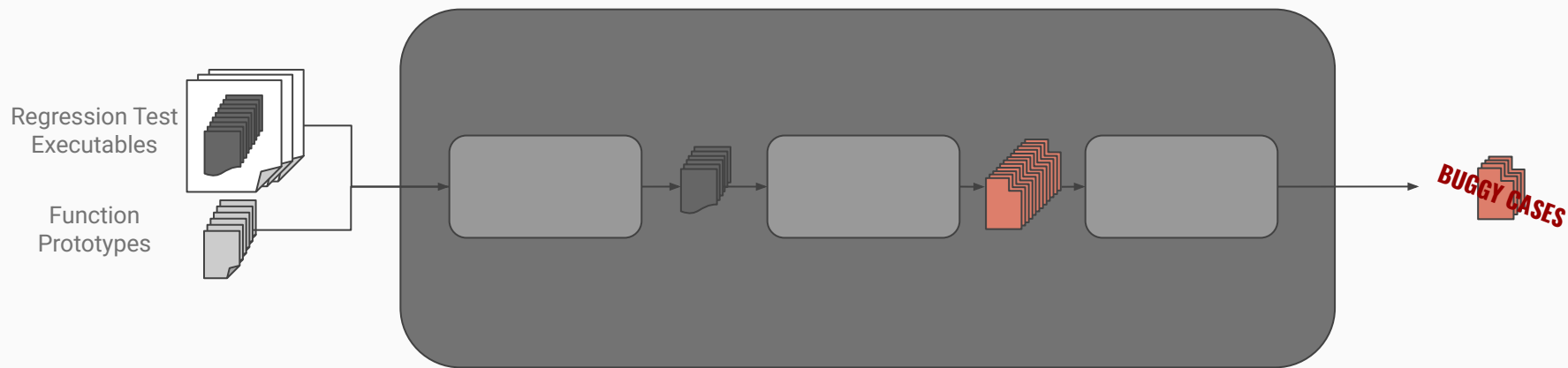
When an Overflow, Divide-by-Zero, or Invalid exception occurs, a **sound exception-handling policy** notifies users by either...

1. The presence of EVs in the output
2. Some library-specific mechanism triggered by checking for their presence at some point during the execution.

This is the policy adopted by the reference LAPACK/BLAS implementations [1].

**-> How can we test this?**





EXCVATE is composed of three components  
that address three key challenges...

# Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

# Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

[1] A. Tran, I. Laguna, G. Gopalakrishnan. "FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs".

[2] E.T. Barr, T. Vo, V. Le, Z. Su. "Automatic Detection of Floating-Point Exceptions".

[3] X. Wu, L. Li, J. Zhang. "Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection".

[4] I. Laguna, G. Gopalakrishnan. "Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization".

# Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

They are designed with the “all exceptions are bugs” mindset; they only seek to trigger exceptions...

[1] A. Tran, I. Laguna, G. Gopalakrishnan. “FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs”.

[2] E.T. Barr, T. Vo, V. Le, Z. Su. “Automatic Detection of Floating-Point Exceptions”.

[3] X. Wu, L. Li, J. Zhang. “Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection”.

[4] I. Laguna, G. Gopalakrishnan. “Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization”.

# Challenge 1: Existing input generation techniques are not well-suited to testing exception handling

Pre-existing works use a variety of input generation techniques to yield floating-point inputs that trigger exceptions.

- e.g., fuzzing [1], symbolic execution [2,3], Bayesian optimization [4]

They are designed with the “all exceptions are bugs” mindset; they only seek to trigger exceptions... but most code should handle exceptions correctly!

- IEEE754 mandates propagation of exceptional-values in most cases
- Developers often do a good job!

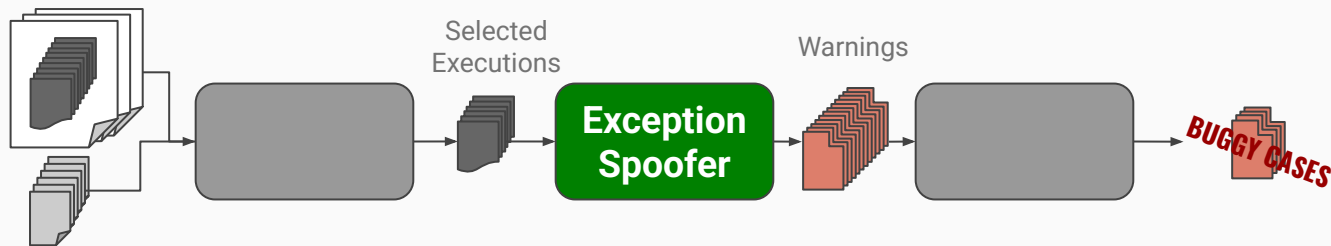
[1] A. Tran, I. Laguna, G. Gopalakrishnan. “FPBoxer: Efficient Input-Generation for Targeting Floating-Point Exceptions in GPU Programs”.

[2] E.T. Barr, T. Vo, V. Le, Z. Su. “Automatic Detection of Floating-Point Exceptions”.

[3] X. Wu, L. Li, J. Zhang. “Symbolic Execution with Value-Range Analysis for Floating-Point Exception Detection”.

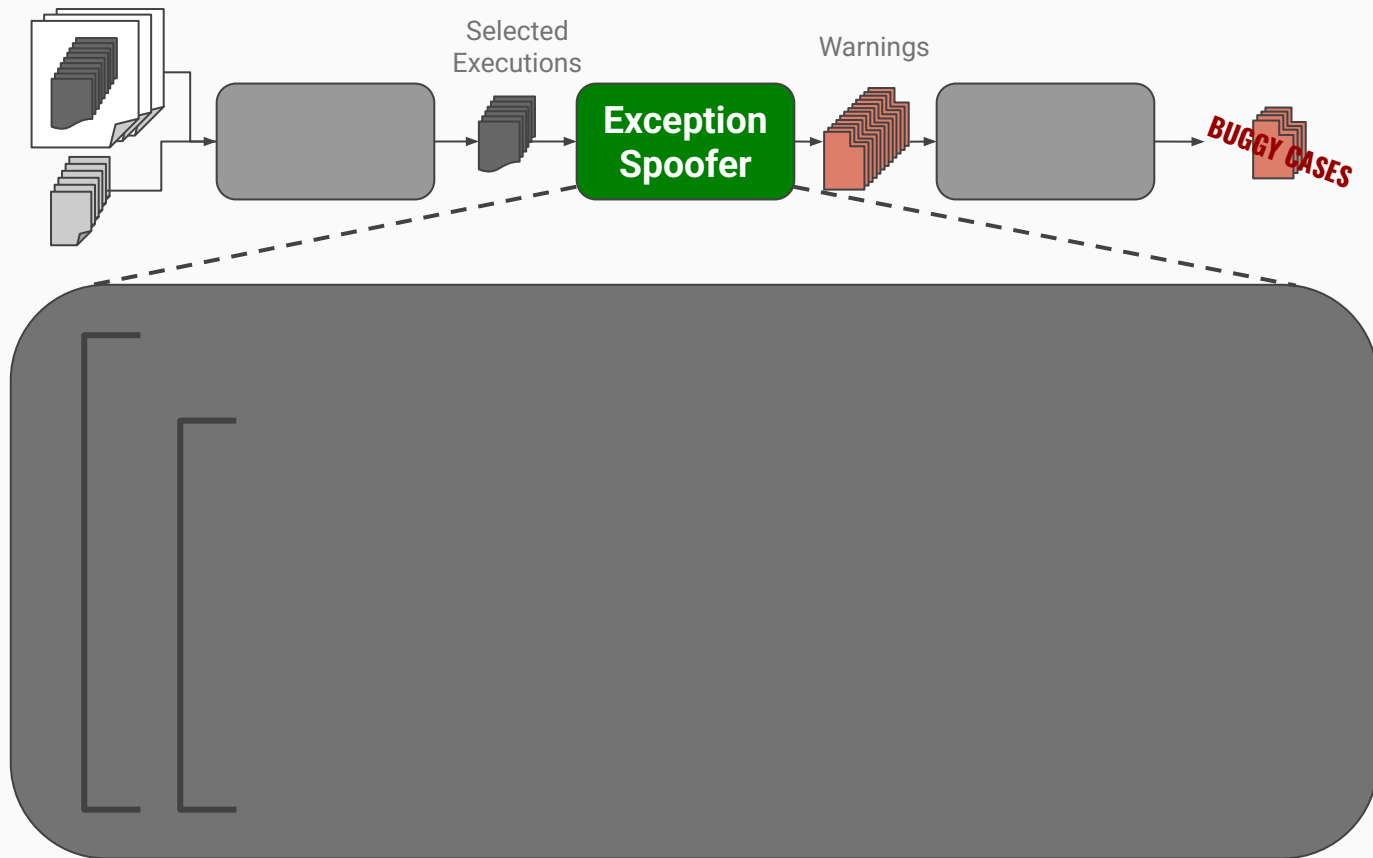
[4] I. Laguna, G. Gopalakrishnan. “Finding Inputs that Trigger Floating-Point Exceptions in GPUs via Bayesian Optimization”.

# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

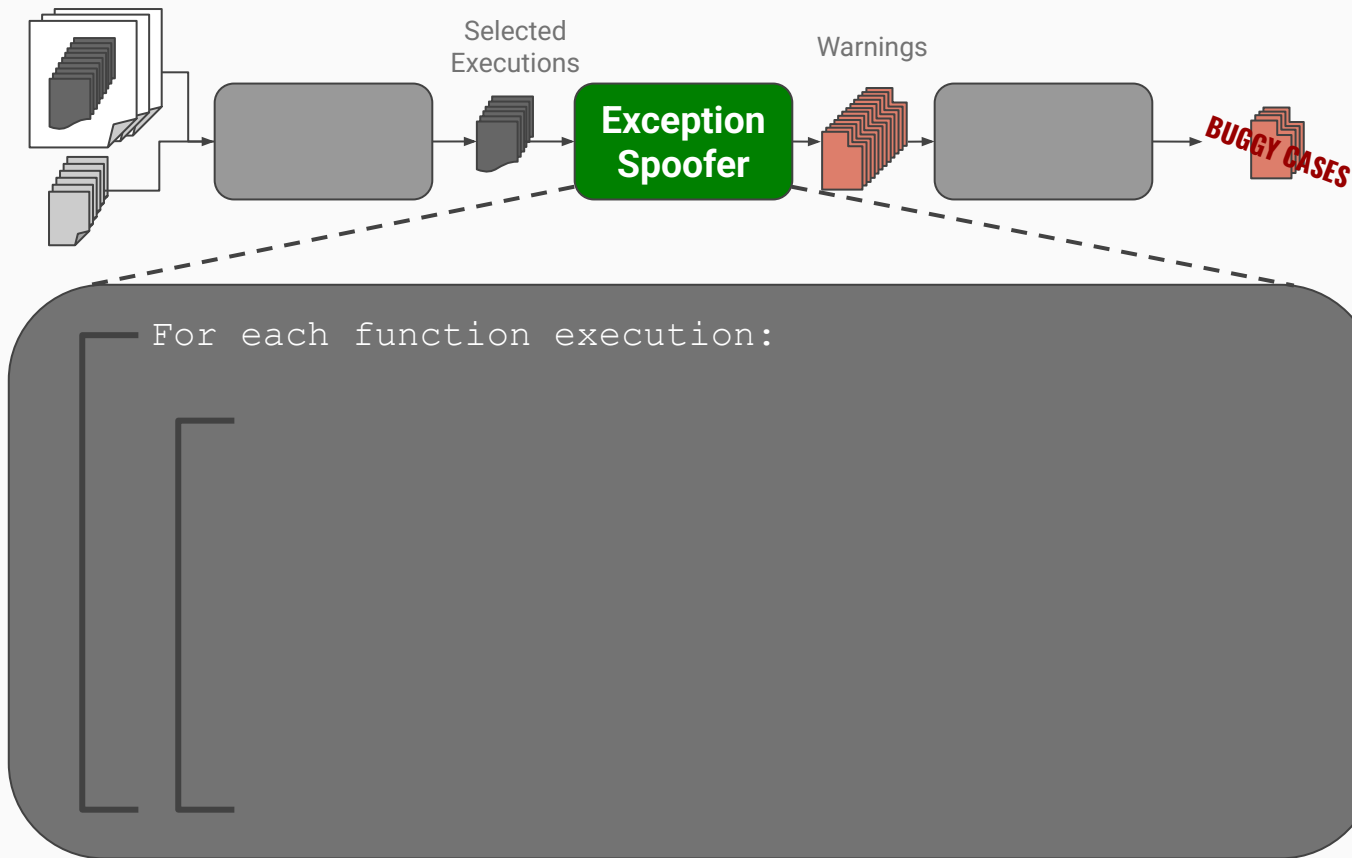




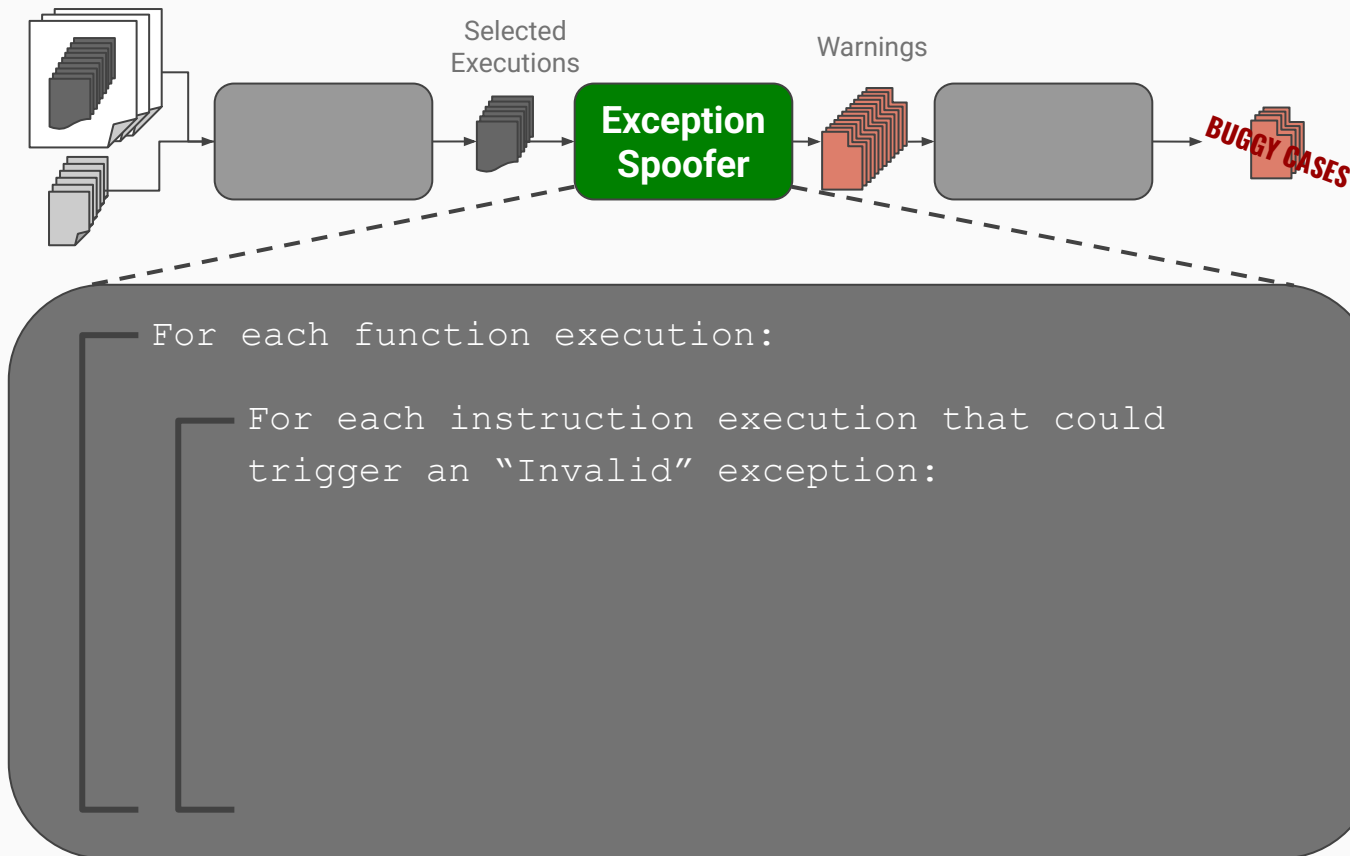
# Approach: spoof exceptions via binary rewriting to cheaply find handling failures



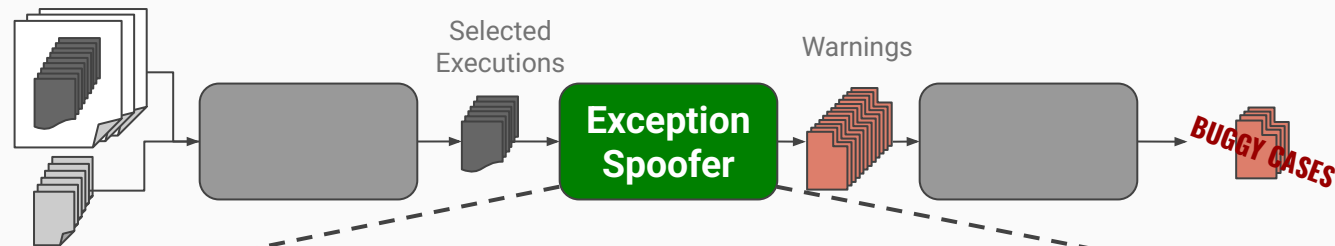
# Approach: spoof exceptions via binary rewriting to cheaply find handling failures



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

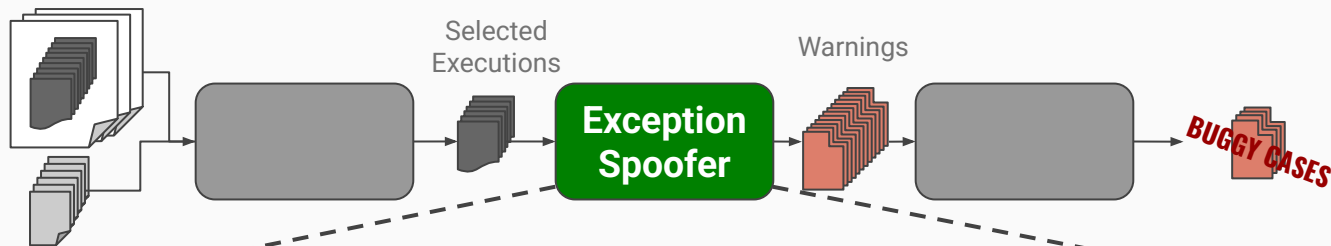


For each function execution:

For each instruction execution that could trigger an "Invalid" exception:

Replay the function execution

# Approach: spoof exceptions via binary rewriting to cheaply find handling failures



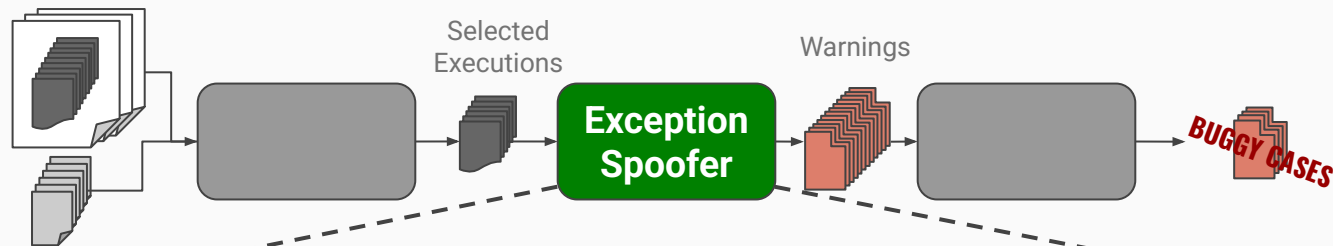
For each function execution:

For each instruction execution that could trigger an "Invalid" exception:

Replay the function execution

Spoof an exception by overwriting the instruction's output with a NaN

# Approach: spoof exceptions via binary rewriting to cheaply find handling failures



For each function execution:

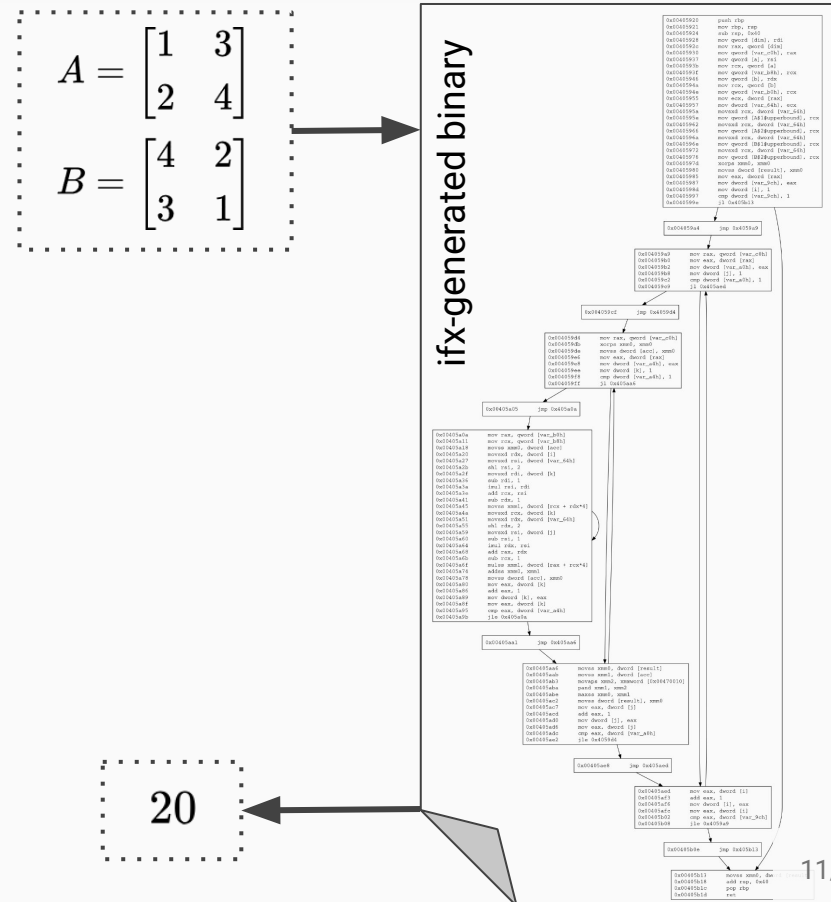
For each instruction execution that could trigger an "Invalid" exception:

Replay the function execution

Spoof an exception by overwriting the instruction's output with a NaN

Raise a warning if handled incorrectly

## Approach: spoof exceptions via binary rewriting to cheaply find handling failures



## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

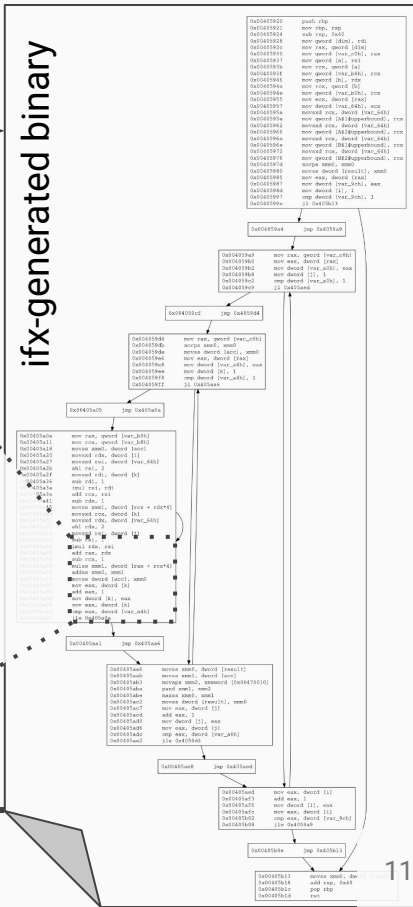
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

20





## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times  $\rightarrow 2 \times 8 = 16$  possible exception sites.

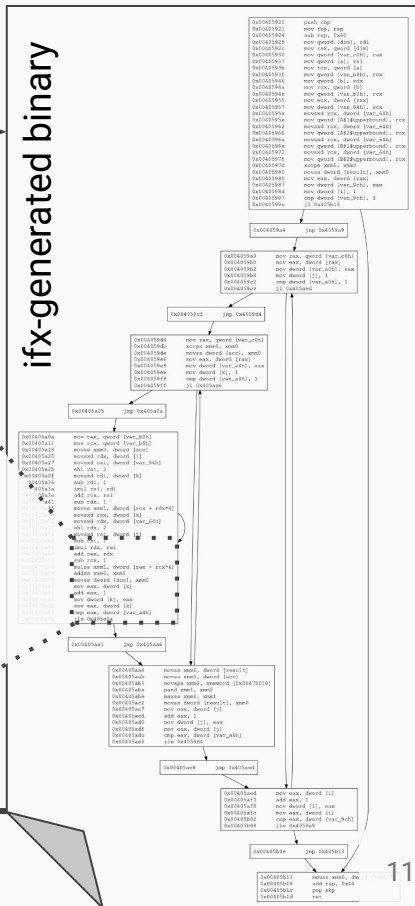
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

20



## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times  $\rightarrow 2 \times 8 = 16$  possible exception sites.

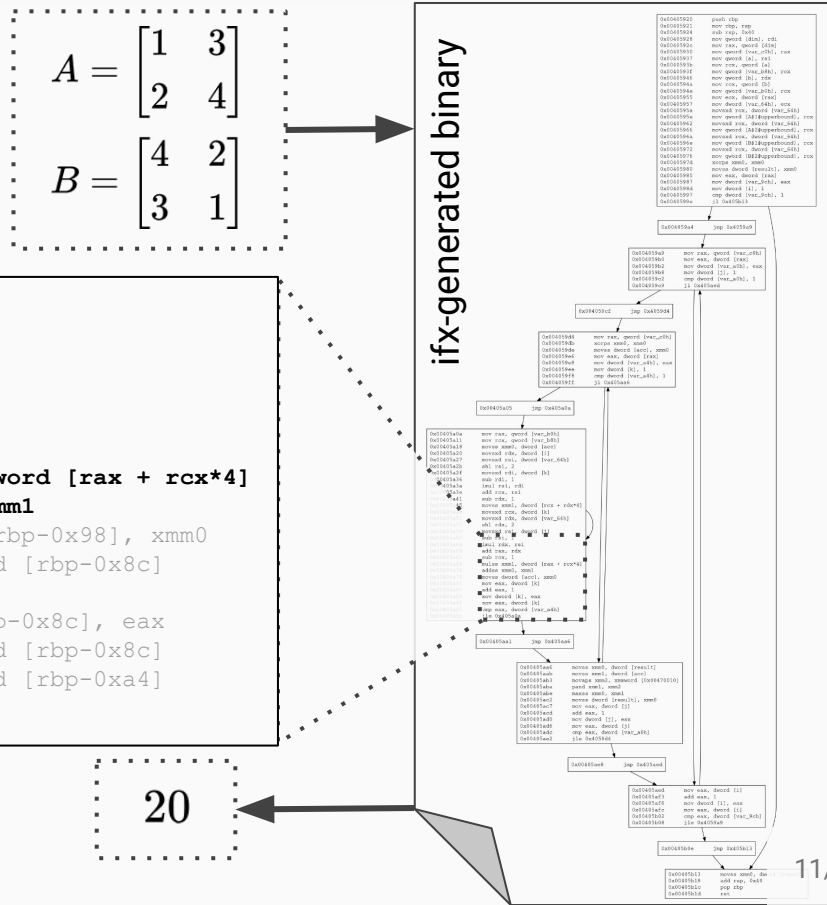
So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

20



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

|     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----|---|---|---|---|---|---|---|---|
| mul |   |   |   |   |   |   |   |   |
| add |   |   |   |   |   |   |   |   |

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
...
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
```

ifx-generated binary

20

# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter "Invalid" exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

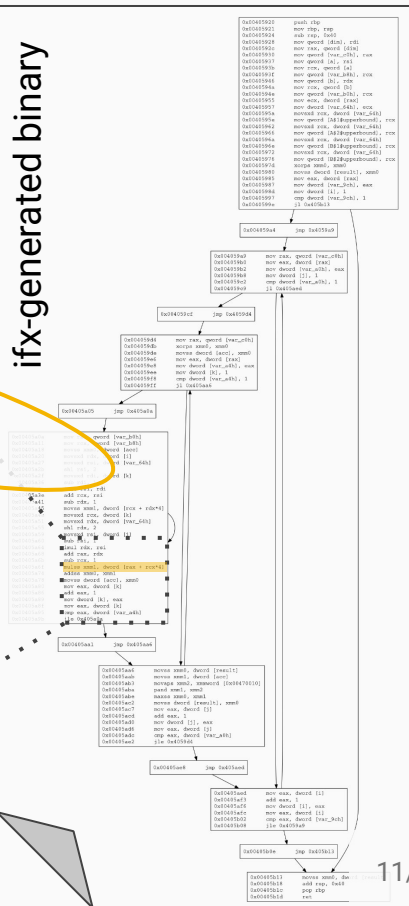
Table of Test Results

|     |    |   |   |   |   |   |   |
|-----|----|---|---|---|---|---|---|
| mul | 20 |   |   |   |   |   |   |
|     |    |   |   |   |   |   |   |
| add |    |   |   |   |   |   |   |
|     |    |   |   |   |   |   |   |
|     | 1  | 2 | 3 | 4 | 5 | 6 | 7 |
|     |    |   |   |   |   |   | 8 |

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN the 1st time it is executed



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter "Invalid" exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

Table of Test Results

|     |    |    |   |   |   |   |   |
|-----|----|----|---|---|---|---|---|
| mul | 20 | 20 |   |   |   |   |   |
|     |    |    |   |   |   |   |   |
| add |    |    |   |   |   |   |   |
|     |    |    |   |   |   |   |   |
|     | 1  | 2  | 3 | 4 | 5 | 6 | 7 |
|     |    |    |   |   |   |   | 8 |

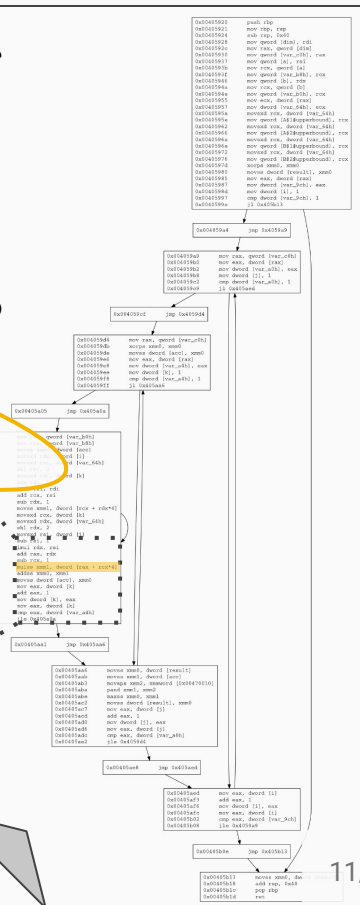
Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN the 2nd time it is executed

ifx-generated binary

```
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
```





# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter "Invalid" exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

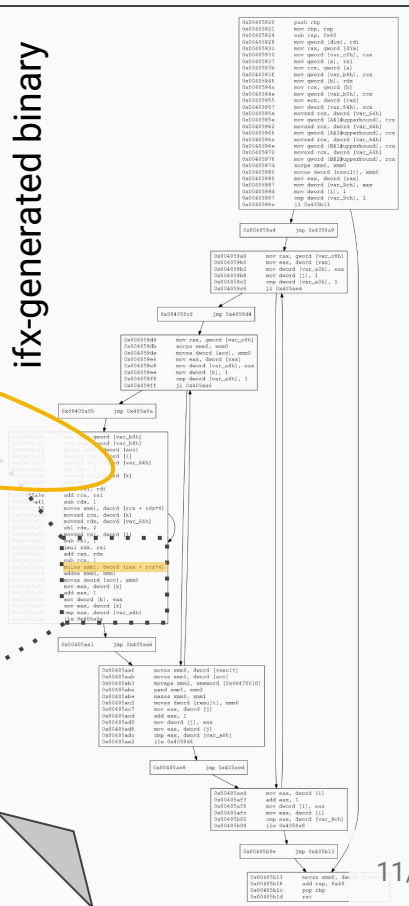
Table of Test Results

|     |    |    |    |    |   |   |   |   |
|-----|----|----|----|----|---|---|---|---|
| mul | 20 | 20 | 20 | 20 |   |   |   |   |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7 | 8 |
| add |    |    |    |    |   |   |   |   |
|     |    |    |    |    |   |   |   |   |

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN the 4th time it is executed







## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times  $\rightarrow 2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

## Table of Test Results

|     |    |    |    |    |   |   |   |   |
|-----|----|----|----|----|---|---|---|---|
| mul | 20 | 20 | 20 | 20 | 8 | 8 |   |   |
| add |    |    |    |    |   |   |   |   |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7 | 8 |

Iteration in which EXCVATE  
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN  
the **6th** time it is executed

```
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx]
addss xmm1, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a
```

# ifx-generated binary



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter "Invalid" exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

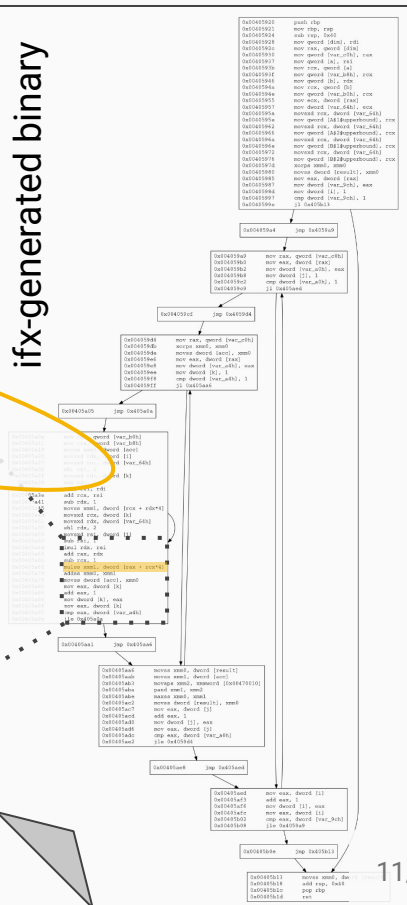
Table of Test Results

|     |    |    |    |    |   |   |     |   |
|-----|----|----|----|----|---|---|-----|---|
| mul | 20 | 20 | 20 | 20 | 8 | 8 | NaN |   |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7   | 8 |
| add |    |    |    |    |   |   |     |   |
|     |    |    |    |    |   |   |     |   |

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN the 7th time it is executed



# Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter "Invalid" exceptions.

These two are in a block that is executed eight times ->  $2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

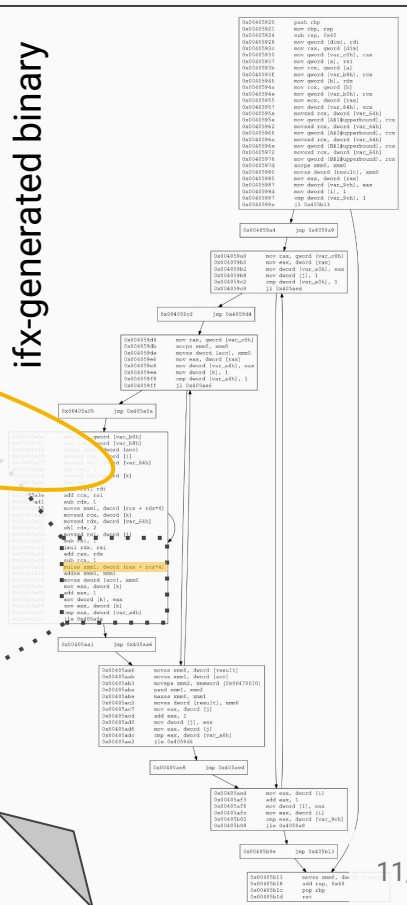
Table of Test Results

|     |    |    |    |    |   |   |     |     |
|-----|----|----|----|----|---|---|-----|-----|
| mul | 20 | 20 | 20 | 20 | 8 | 8 | NaN | NaN |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7   | 8   |
| add |    |    |    |    |   |   |     |     |
|     |    |    |    |    |   |   |     |     |

Iteration in which EXCVATE spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

Overwrite output with NaN the **8th** time it is executed



## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times  $\rightarrow 2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

## Table of Test Results

|     |    |    |    |    |   |   |     |     |
|-----|----|----|----|----|---|---|-----|-----|
| mul | 20 | 20 | 20 | 20 | 8 | 8 | NaN | NaN |
| add |    |    |    |    |   |   |     |     |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7   | 8   |

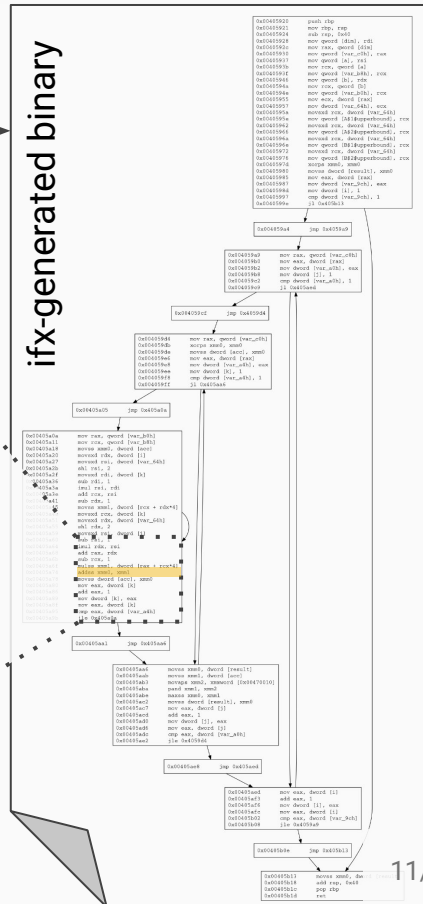
Iteration in which EXCVATE  
spoofs an exception

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:
sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```



## Approach: spoof exceptions via binary rewriting to cheaply find handling failures

There are only two instructions in the ifx binary that could encounter “Invalid” exceptions.

These two are in a block that is executed eight times  $\rightarrow 2 \times 8 = 16$  possible exception sites.

So, replay the function execution 16 times, **spoofing** a different Invalid exception each time.

## Table of Test Results

|     |    |    |    |    |   |   |     |     |
|-----|----|----|----|----|---|---|-----|-----|
| mul | 20 | 20 | 20 | 20 | 8 | 8 | NaN | NaN |
| add | 20 | 20 | 20 | 20 | 8 | 8 | NaN | NaN |
|     | 1  | 2  | 3  | 4  | 5 | 6 | 7   | 8   |

Iteration in which EXCVATE  
spoofs an exception

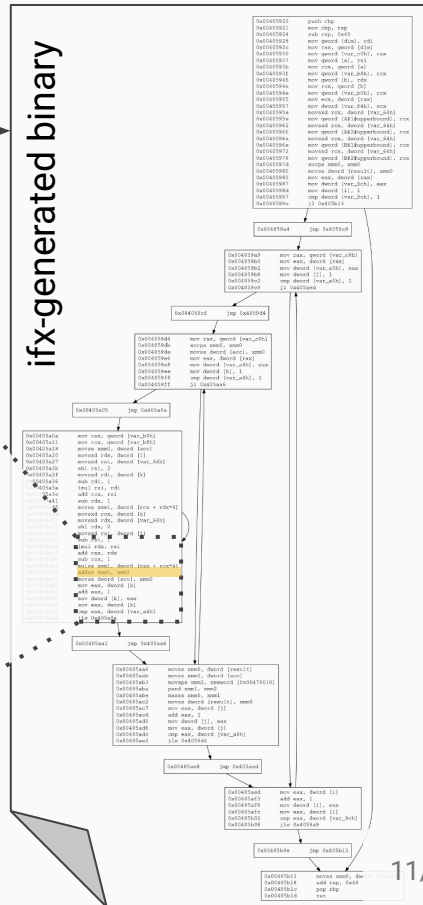
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

sub rsi, 1
imul rdx, rsi
add rax, rdx
sub rcx, 1
mulss xmm1, dword [rax + rcx*4]
addss xmm0, xmm1
movss dword [rbp-0x98], xmm0
mov eax, dword [rbp-0x8c]
add eax, 1
mov dword [rbp-0x8c], eax
mov eax, dword [rbp-0x8c]
cmp eax, dword [rbp-0xa4]
jle 0x405a0a

```

# ifx-generated binary



Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

## Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

Warnings can originate from spoofed Invalid exceptions that are not possible.

- e.g., `if ( isfinite(x) && isfinite(y) ) z = x*y;` -> Invalid exception is impossible

## Challenge 2: How do we “reify” a spoofed exception that results in an exception-handling failure?

Warnings can originate from spoofed Invalid exceptions that are not possible.

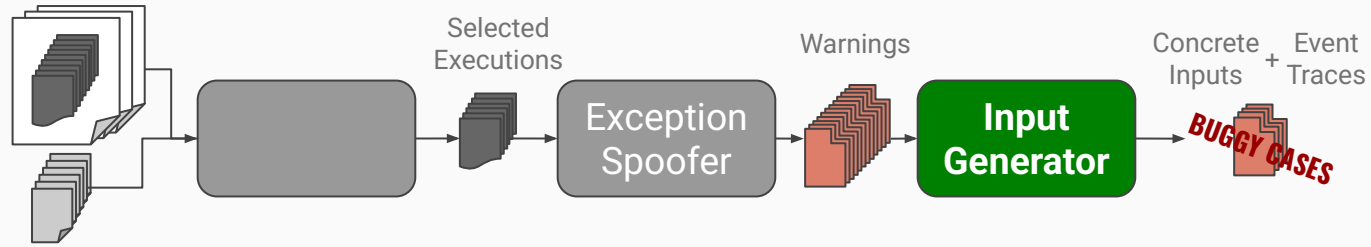
○ e.g., `if ( isfinite(x) && isfinite(y) ) z = x*y;` -> Invalid exception is impossible

For a true buggy case, EXCVATE must create an input that:

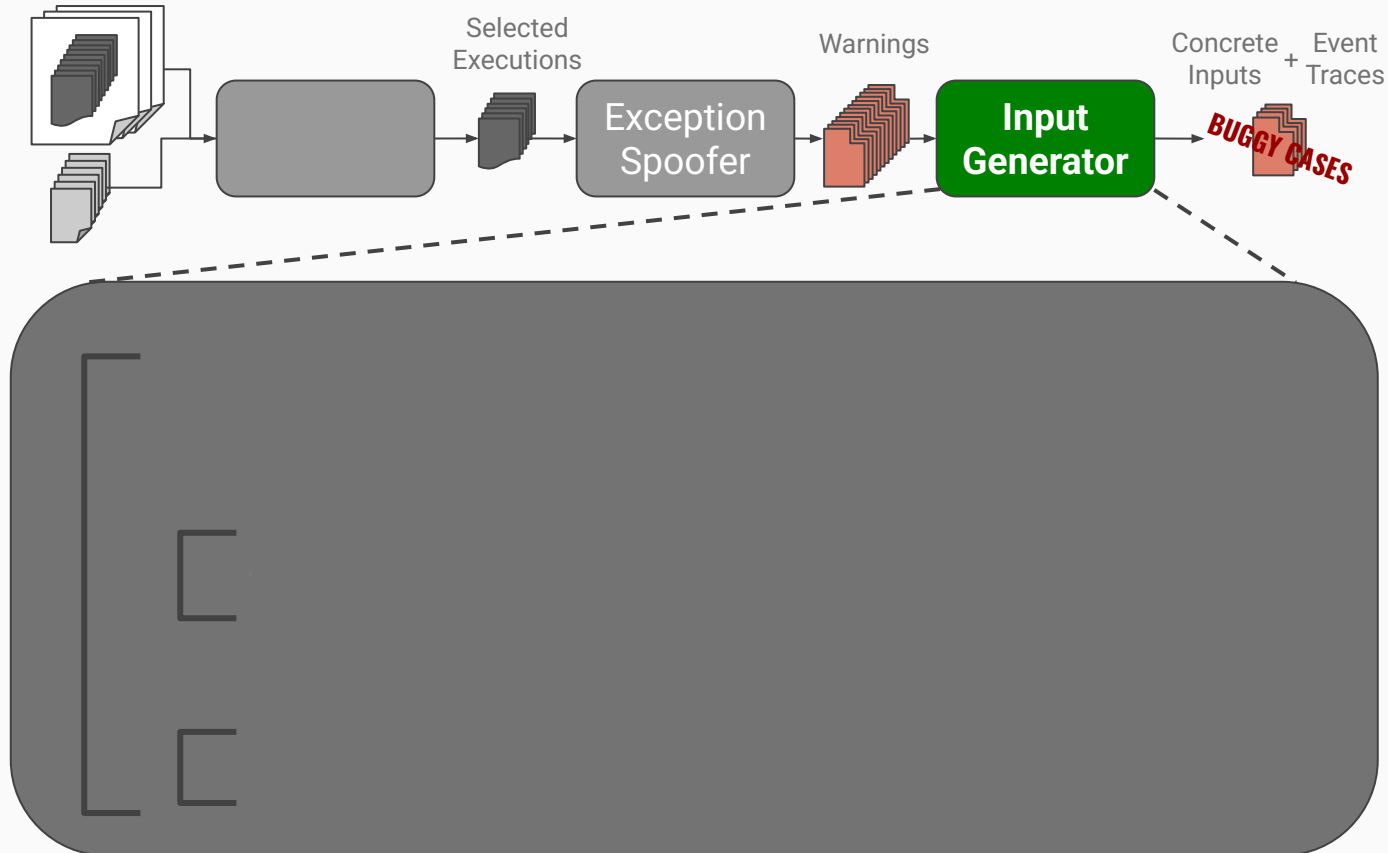
1. triggers the spoofed Invalid exception
2. preserves the control flow that resulted in failed exception handling



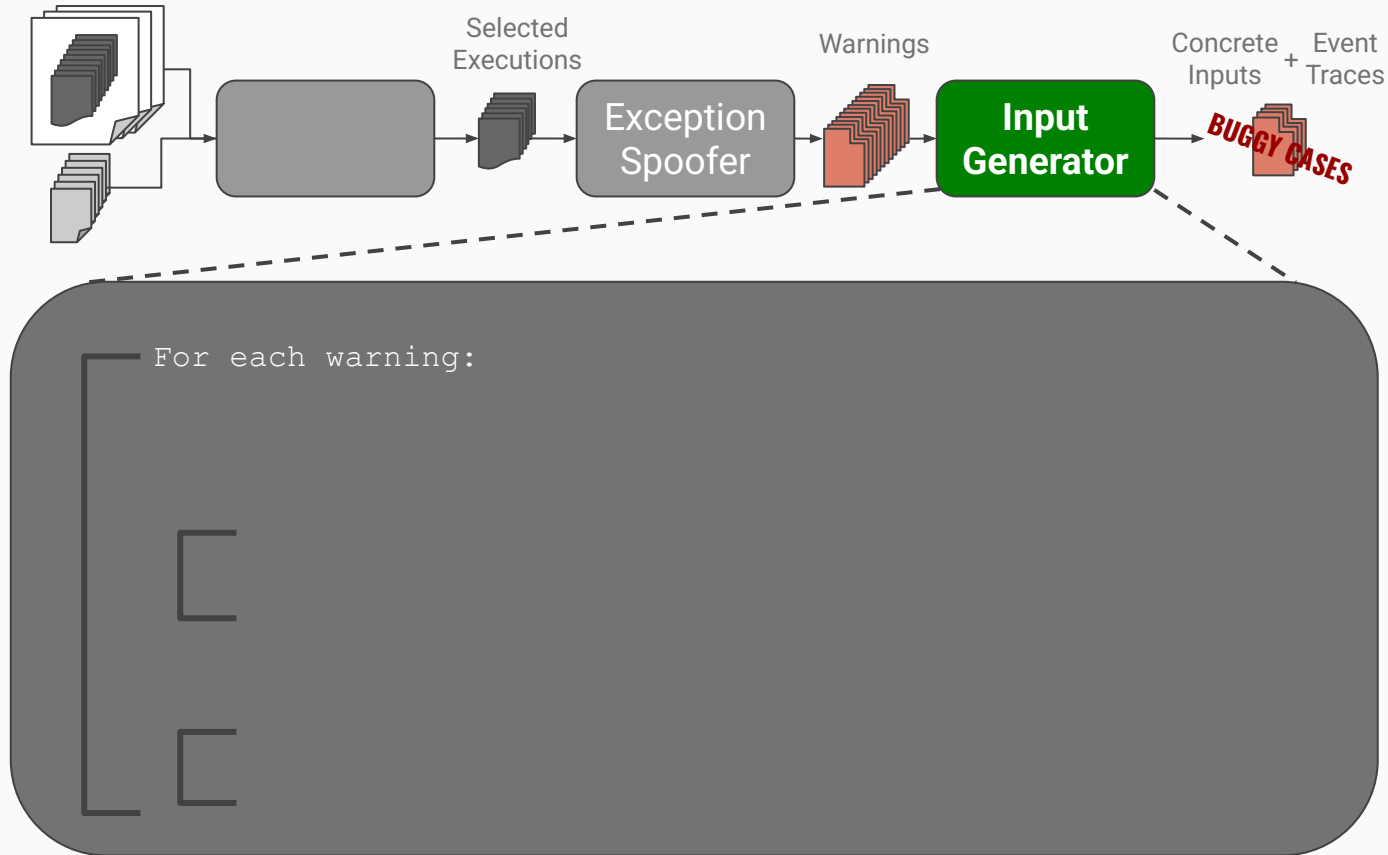
# Approach: encode the desired behavior (control flow + exception) into an SMT query



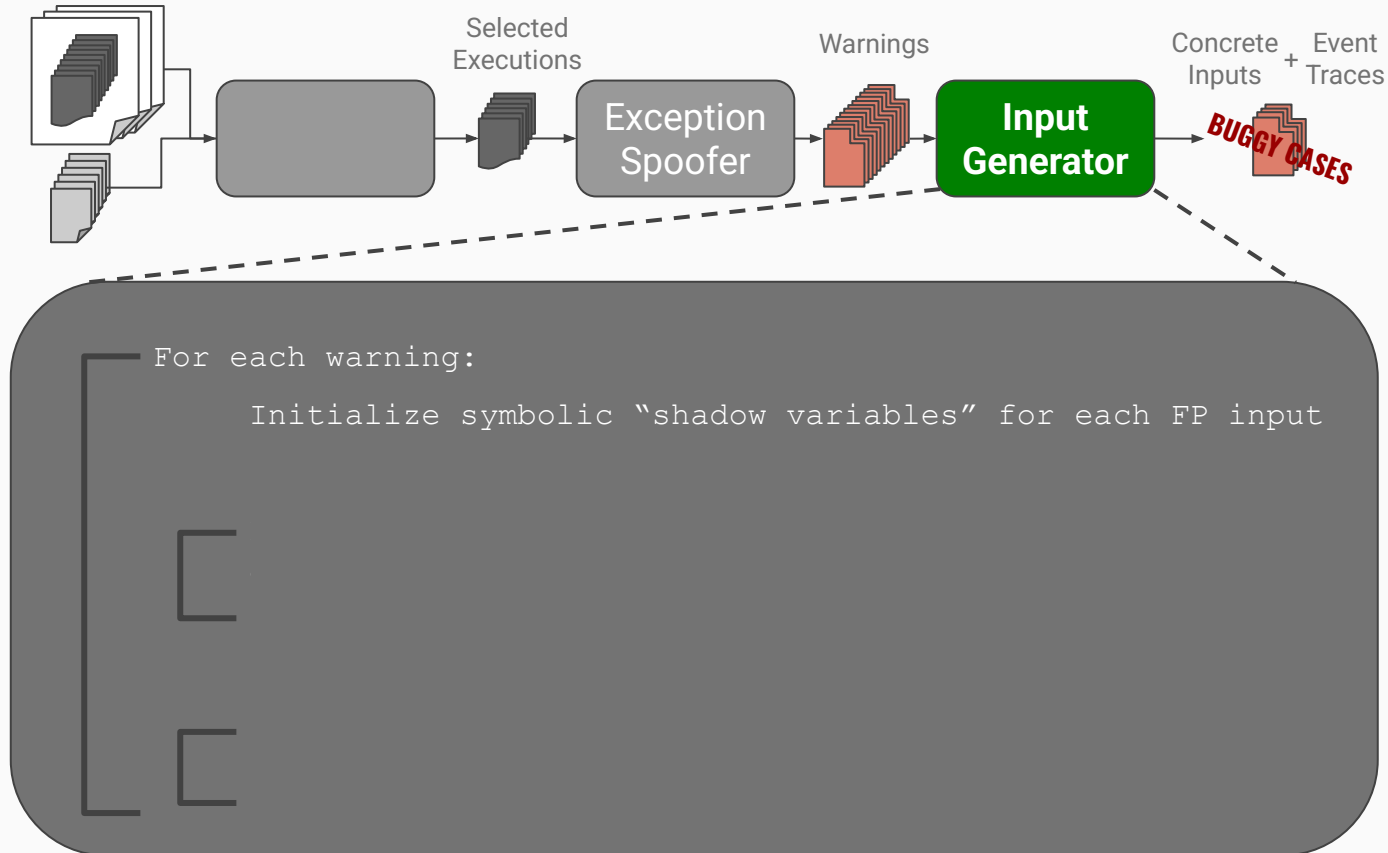
# Approach: encode the desired behavior (control flow + exception) into an SMT query



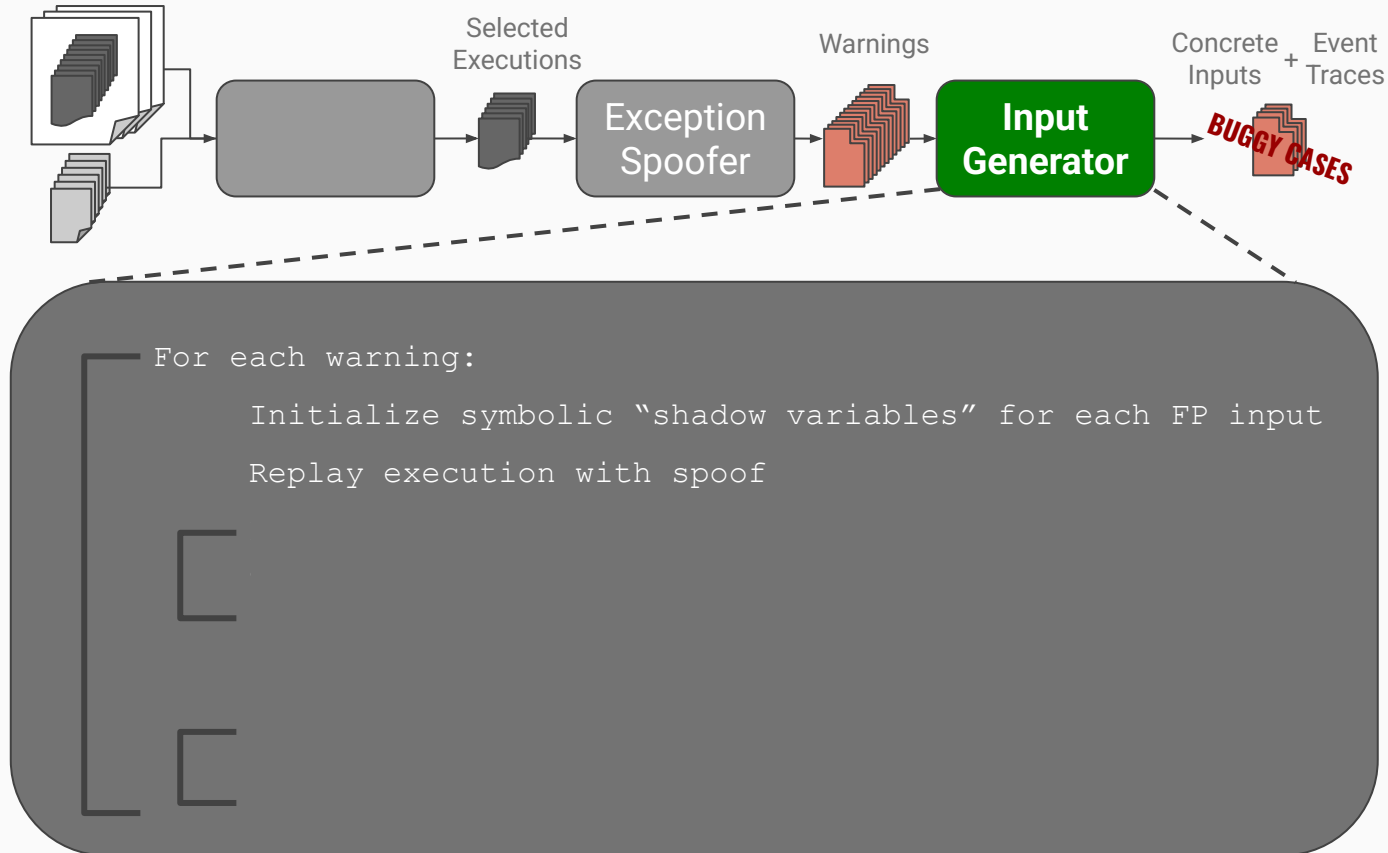
# Approach: encode the desired behavior (control flow + exception) into an SMT query



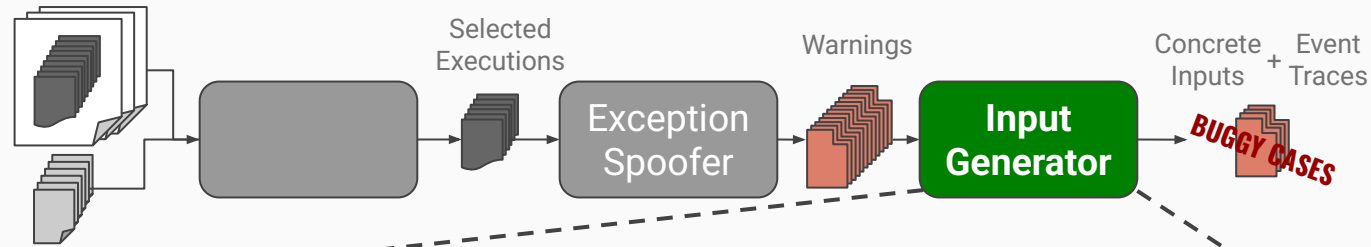
# Approach: encode the desired behavior (control flow + exception) into an SMT query



# Approach: encode the desired behavior (control flow + exception) into an SMT query



# Approach: encode the desired behavior (control flow + exception) into an SMT query



For each warning:

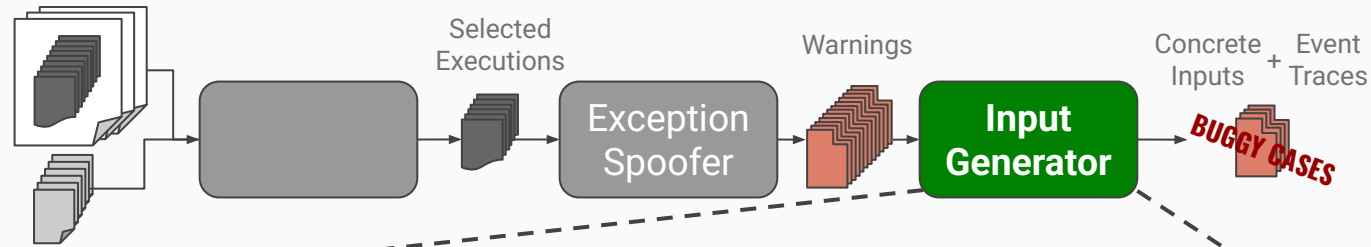
- Initialize symbolic "shadow variables" for each FP input

- Replay execution with spoof

- For each instruction execution whose operands have associated symbolic shadow variables:

  - Add a constraint to the SMT query

# Approach: encode the desired behavior (control flow + exception) into an SMT query



For each warning:

- Initialize symbolic "shadow variables" for each FP input

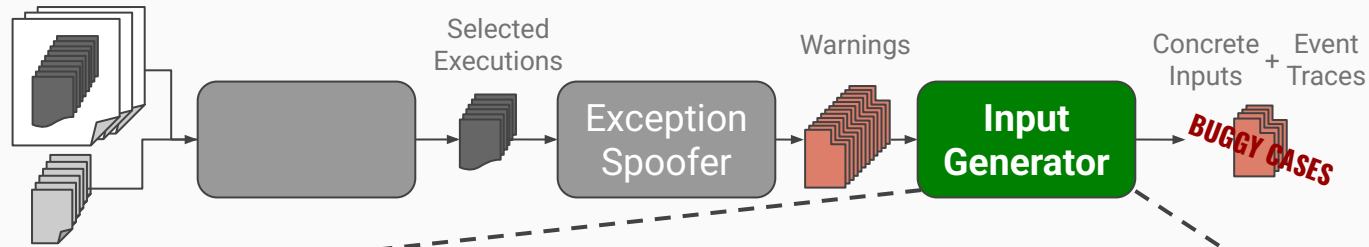
- Replay execution with spoof

- For each instruction execution whose operands have associated symbolic shadow variables:

  - Add a constraint to the SMT query

- Give query to an SMT solver

# Approach: encode the desired behavior (control flow + exception) into an SMT query



For each warning:

- Initialize symbolic "shadow variables" for each FP input

- Replay execution with spoof

- For each instruction execution whose operands have associated symbolic shadow variables:

  - Add a constraint to the SMT query

- Give query to an SMT solver

- If satisfiable:

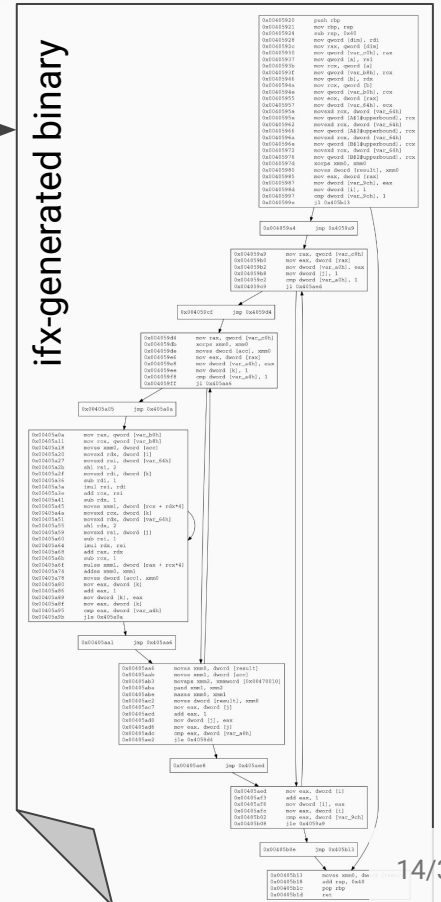
  - Run program with generated input



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$



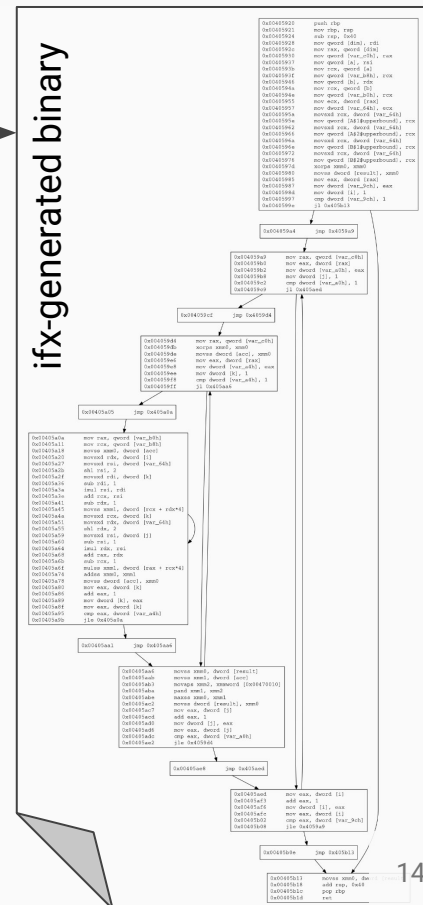
**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

$$A_{sym} = \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$

$$B_{sym} = \begin{bmatrix} b_0 & b_2 \\ b_1 & b_3 \end{bmatrix}$$



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

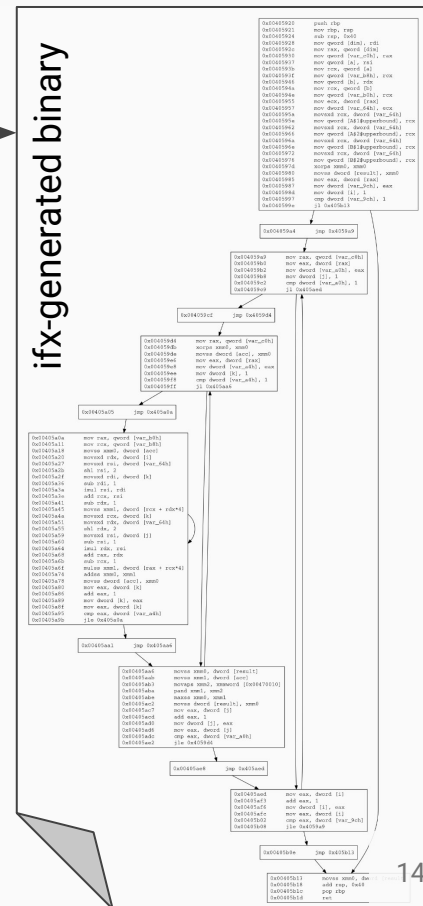
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

$$A_{sym} = \begin{bmatrix} a_0 & a_2 \\ a_1 & a_3 \end{bmatrix}$$

$$B_{sym} = \begin{bmatrix} b_0 & b_2 \\ b_1 & b_3 \end{bmatrix}$$

# Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```





**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,
    end do
    result = max(result, abs(acc))
  end do
end do
```

# Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

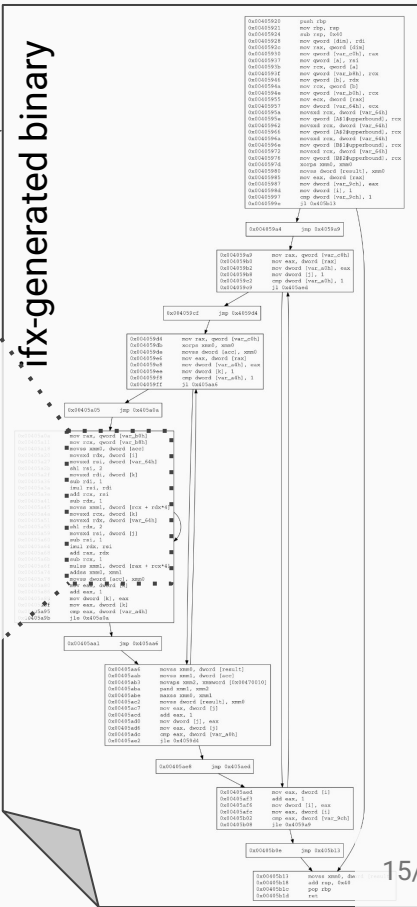
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]
:
movss xmm1, dword ptr [rcx+rdx*4]
:
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0

```

## ifx-generated binary



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,
    end do
    result = max(result, abs(acc))
  end do
end do
```

## Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

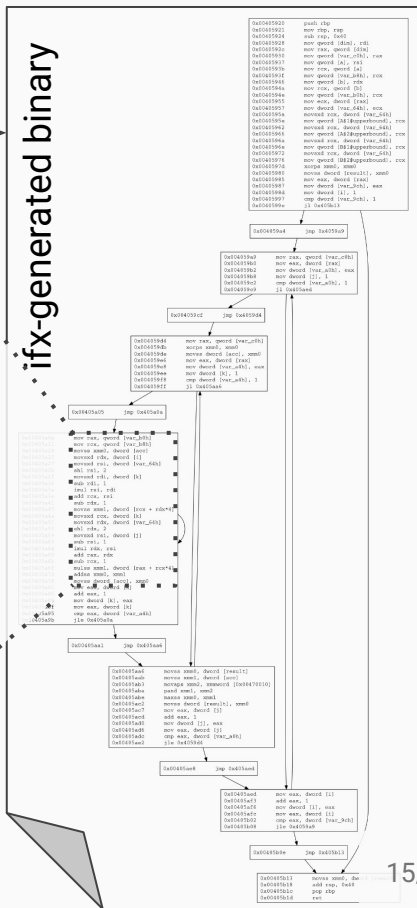
```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]
:
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
:
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0

```

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

ifx-generated binary



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,
    end do
    result = max(result, abs(acc))
  end do
end do
```

# Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

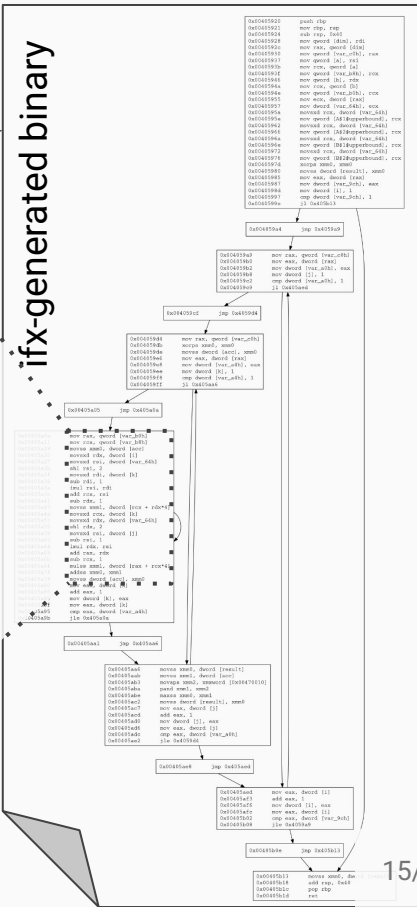
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]
:
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
:
:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:

```

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
result = 0.0
do i = 1, dim
  do j = 1, dim
    acc = 0.0
    do k = 1, dim
      acc = acc + A(i,k) * B(k,j)
    end do
    result = max(result, abs(acc))
  end do
end do
```

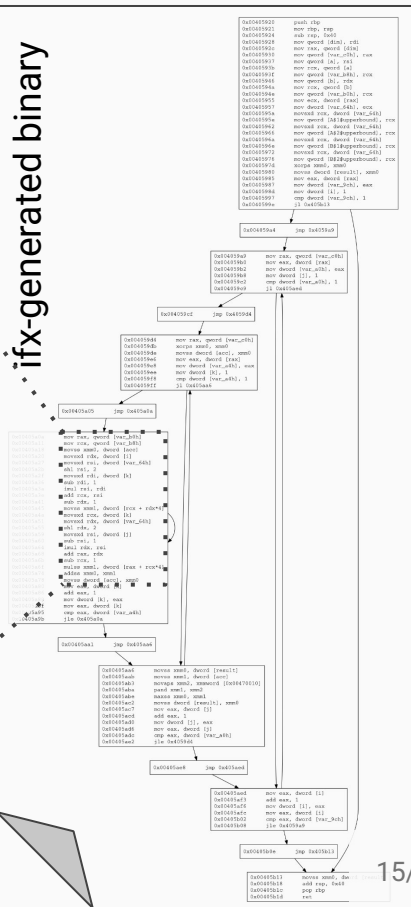
Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
...
... -> accumulate and save acc
```

ifx-generated binary





**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

ifx-generated binary

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

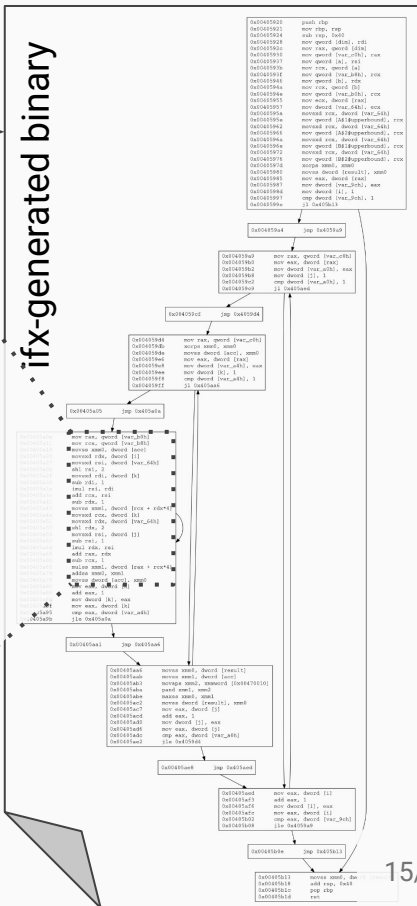
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

## Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
...
... -> accumulate and save acc
```

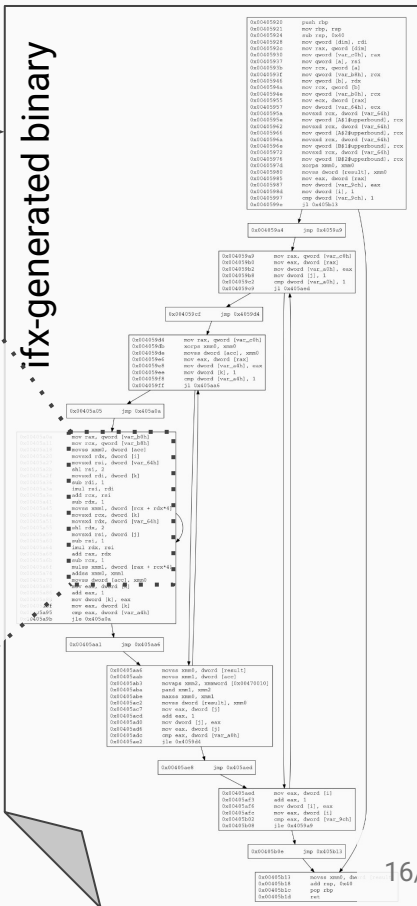
Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

Read Operands

0x7ffef9945e28

ifx-generated binary



## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

## Active Symbols

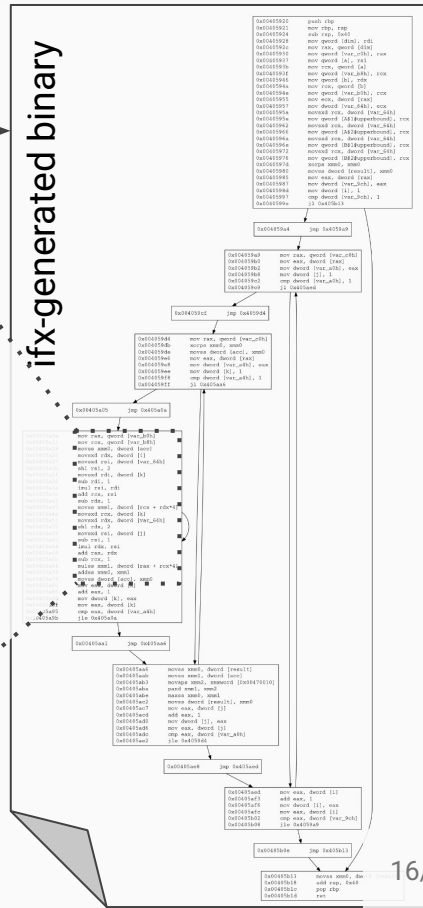
```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

## Read Operands

Read Operands

**NOT IN ACTIVE SYMBOLS**

0x39945e28



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

ifx-generated binary

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

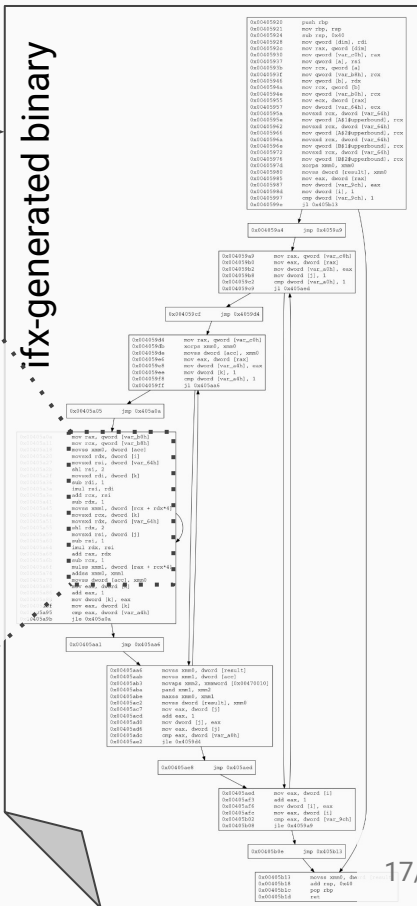
:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

## Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

## ifx-generated binary

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

## Read Operands

0x7f9dea0970a4

## Active Symbols

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

[illegible]

```
0x004859a4      jmp 0x0059a9
0059a9      mov rax, qword [var_0fh]
0059b0      mov eax, dword [rax]
0059b2      mov dword [var_0fh], eax
0059b4      mov dword [i], 1
0059c2      cmp dword [var_0fh], 1
0059c9      jz 0x004859ad
```

```

4 mov rax, dword [var_a0h]
5 scasd rax, xmm0
6 movs dword [sec], xmm0
7 mov rax, dword [rax]
8 mov dword [var_a0h], rax
9 mov dword [i], 1
10 cmp dword [var_a0h], 1
11 jf 0x405aa6

```

```

0x04005a05      jmp 0x04005a05
0x04005a06      mov rax, qword [var_18h]
0x04005a07      mov rax, qword [var_18h]
0x04005a08      movss xmm0, dword [eax]
0x04005a09      movss rdx, dword [1]
0x04005a0a      movss rsi, dword [var_64]
0x04005a0b      shl rax, 2
0x04005a0c      movss rdi, dword [k]
0x04005a0d      sub rdi, 1
0x04005a0e      imul rsi, rdi
0x04005a0f      add rax, rsi

```

```

mov rax, dword [var_64]
mov rax, dword [k]
shl rax, 2
mov rax, dword [i]
sub rax, 1
leal rax, rax
add rax, rdx
mov rax, 1
mul rax, dword [rax * 4]
add rax, rax
mov rax, dword [acc], rax
add rax, 1
mov dword [k], rax

```

```

jmp 0x0405a6
0x0405a1: jmp 0x0405a1
0x0405a4: 0x0405a4
0x0405a5: 0x0405a5
0x0405a6: 0x0405a6
0x0405a7: 0x0405a7
0x0405a8: 0x0405a8
0x0405a9: 0x0405a9
0x0405aa: 0x0405aa
0x0405ab: 0x0405ab
0x0405ac: 0x0405ac
0x0405ad: 0x0405ad
0x0405ae: 0x0405ae
0x0405af: 0x0405af
0x0405b0: 0x0405b0
0x0405b1: 0x0405b1
0x0405b2: 0x0405b2
0x0405b3: 0x0405b3
0x0405b4: 0x0405b4
0x0405b5: 0x0405b5
0x0405b6: 0x0405b6
0x0405b7: 0x0405b7
0x0405b8: 0x0405b8
0x0405b9: 0x0405b9
0x0405ba: 0x0405ba
0x0405bb: 0x0405bb
0x0405bc: 0x0405bc
0x0405bd: 0x0405bd
0x0405be: 0x0405be
0x0405bf: 0x0405bf
0x0405c0: 0x0405c0
0x0405c1: 0x0405c1
0x0405c2: 0x0405c2
0x0405c3: 0x0405c3
0x0405c4: 0x0405c4
0x0405c5: 0x0405c5
0x0405c6: 0x0405c6
0x0405c7: 0x0405c7
0x0405c8: 0x0405c8
0x0405c9: 0x0405c9
0x0405ca: 0x0405ca
0x0405cb: 0x0405cb
0x0405cc: 0x0405cc
0x0405cd: 0x0405cd
0x0405ce: 0x0405ce
0x0405cf: 0x0405cf
0x0405d0: 0x0405d0
0x0405d1: 0x0405d1
0x0405d2: 0x0405d2
0x0405d3: 0x0405d3
0x0405d4: 0x0405d4
0x0405d5: 0x0405d5
0x0405d6: 0x0405d6
0x0405d7: 0x0405d7
0x0405d8: 0x0405d8
0x0405d9: 0x0405d9
0x0405da: 0x0405da
0x0405db: 0x0405db
0x0405dc: 0x0405dc
0x0405dd: 0x0405dd
0x0405de: 0x0405de
0x0405df: 0x0405df
0x0405e0: 0x0405e0
0x0405e1: 0x0405e1
0x0405e2: 0x0405e2
0x0405e3: 0x0405e3
0x0405e4: 0x0405e4
0x0405e5: 0x0405e5
0x0405e6: 0x0405e6
0x0405e7: 0x0405e7
0x0405e8: 0x0405e8
0x0405e9: 0x0405e9
0x0405ea: 0x0405ea
0x0405eb: 0x0405eb
0x0405ec: 0x0405ec
0x0405ed: 0x0405ed
0x0405ee: 0x0405ee
0x0405ef: 0x0405ef
0x0405f0: 0x0405f0
0x0405f1: 0x0405f1
0x0405f2: 0x0405f2
0x0405f3: 0x0405f3
0x0405f4: 0x0405f4
0x0405f5: 0x0405f5
0x0405f6: 0x0405f6
0x0405f7: 0x0405f7
0x0405f8: 0x0405f8
0x0405f9: 0x0405f9
0x0405fa: 0x0405fa
0x0405fb: 0x0405fb
0x0405fc: 0x0405fc
0x0405fd: 0x0405fd
0x0405fe: 0x0405fe
0x0405ff: 0x0405ff
0x040600: 0x040600
0x040601: 0x040601
0x040602: 0x040602
0x040603: 0x040603
0x040604: 0x040604
0x040605: 0x040605
0x040606: 0x040606
0x040607: 0x040607
0x040608: 0x040608
0x040609: 0x040609
0x04060a: 0x04060a
0x04060b: 0x04060b
0x04060c: 0x04060c
0x04060d: 0x04060d
0x04060e: 0x04060e
0x04060f: 0x04060f
0x040610: 0x040610
0x040611: 0x040611
0x040612: 0x040612
0x040613: 0x040613
0x040614: 0x040614
0x040615: 0x040615
0x040616: 0x040616
0x040617: 0x040617
0x040618: 0x040618
0x040619: 0x040619
0x04061a: 0x04061a
0x04061b: 0x04061b
0x04061c: 0x04061c
0x04061d: 0x04061d
0x04061e: 0x04061e
0x04061f: 0x04061f
0x040620: 0x040620
0x040621: 0x040621
0x040622: 0x040622
0x040623: 0x040623
0x040624: 0x040624
0x040625: 0x040625
0x040626: 0x040626
0x040627: 0x040627
0x040628: 0x040628
0x040629: 0x040629
0x04062a: 0x04062a
0x04062b: 0x04062b
0x04062c: 0x04062c
0x04062d: 0x04062d
0x04062e: 0x04062e
0x04062f: 0x04062f
0x040630: 0x040630
0x040631: 0x040631
0x040632: 0x040632
0x040633: 0x040633
0x040634: 0x040634
0x040635: 0x040635
0x040636: 0x040636
0x040637: 0x040637
0x040638: 0x040638
0x040639: 0x040639
0x04063a: 0x04063a
0x04063b: 0x04063b
0x04063c: 0x04063c
0x04063d: 0x04063d
0x04063e: 0x04063e
0x04063f: 0x04063f
0x040640: 0x040640
0x040641: 0x040641
0x040642: 0x040642
0x040643: 0x040643
0x040644: 0x040644
0x040645: 0x040645
0x040646: 0x040646
0x040647: 0x040647
0x040648: 0x040648
0x040649: 0x040649
0x04064a: 0x04064a
0x04064b: 0x04064b
0x04064c: 0x04064c
0x04064d: 0x04064d
0x04064e: 0x04064e
0x04064f: 0x04064f
0x040650: 0x040650
0x040651: 0x040651
0x040652: 0x040652
0x040653: 0x040653
0x040654: 0x040654
0x040655: 0x040655
0x040656: 0x040656
0x040657: 0x040657
0x040658: 0x040658
0x040659: 0x040659
0x04065a: 0x04065a
0x04065b: 0x04065b
0x04065c: 0x04065c
0x04065d: 0x04065d
0x04065e: 0x04065e
0x04065f: 0x04065f
0x040660: 0x040660
0x040661: 0x040661
0x040662: 0x040662
0x040663: 0x040663
0x040664: 0x040664
0x040665: 0x040665
0x040666: 0x040666
0x040667: 0x040667
0x040668: 0x040668
0x040669: 0x040669
0x04066a: 0x04066a
0x04066b: 0x04066b
0x04066c: 0x04066c
0x04066d: 0x04066d
0x04066e: 0x04066e
0x04066f: 0x04066f
0x040670: 0x040670
0x040671: 0x040671
0x040672: 0x040672
0x040673: 0x040673
0x040674: 0x040674
0x040675: 0x040675
0x040676: 0x040676
0x040677: 0x040677
0x040678: 0x040678
0x040679: 0x040679
0x04067a: 0x04067a
0x04067b: 0x04067b
0x04067c: 0x04067c
0x04067d: 0
```

```
mov eax, dword [j]
cmp eax, dword [var_addr]
jle 0x0059d6
```

0x0059d8 jmp 0x0059e8

```
0x0059e8 mov eax, dword [var_addr]
0x0059ed mov ebx, dword [var_addr]
0x0059f2 mov ecx, dword [var_addr]
0x0059f9 mov edx, dword [var_addr]
```

```

0x00405b0e      jmp 0x005b13
0x00405b13      movs word, [0x00405b18]
0x00405b18      add  esp, 0x00405b1c
0x00405b1c      pop  ebp
0x00405b1d      ret

```

## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

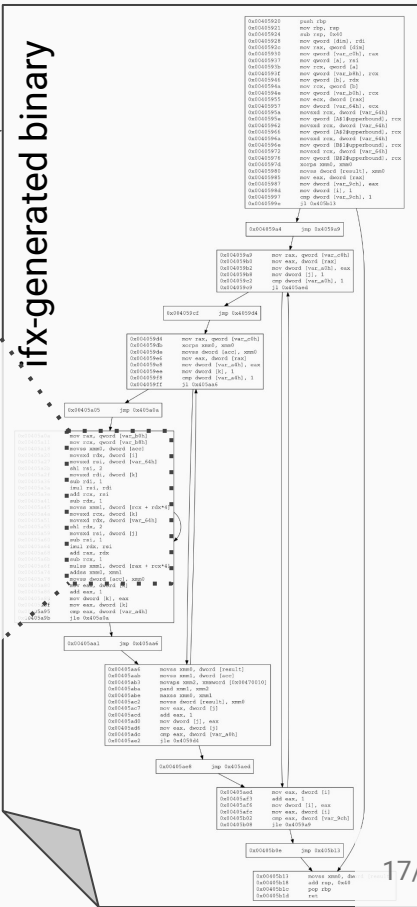
## Active Symbols

0x7f9dea0970a4

## Read Operands

0x7f9dea0970a4

## ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = xmm1_0 0x7f9dea0970a4_0 )
```

SMT  
query

Active Symbols

0x7f9dea0970a4

0x7f9dea0970a8  
0x7f9dea0970ac  
0x7f9dea0970b0  
0x7f9dea0970b4  
0x7f9dea0970b8  
0x7f9dea0970bc  
0x7f9dea0970c0

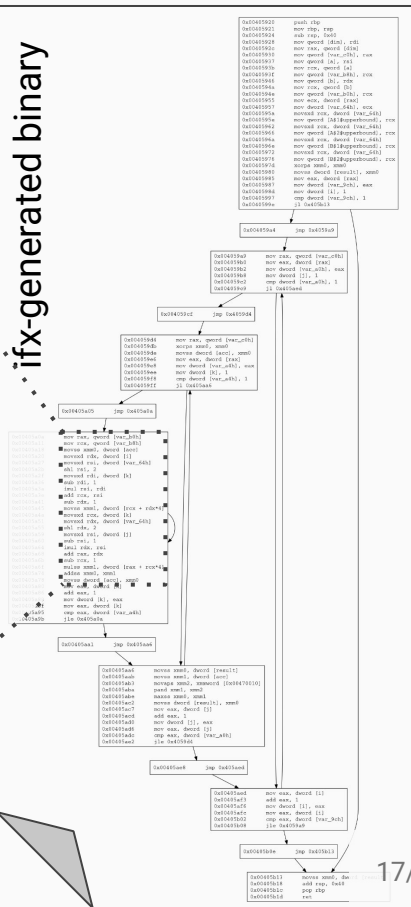
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
... -> accumulate and save acc
```

Read Operands

0x7f9dea0970a4

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = xmm1_0 0x7f9dea0970a4_0 )
```

SMT  
query

Active Symbols

```
xmm1
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

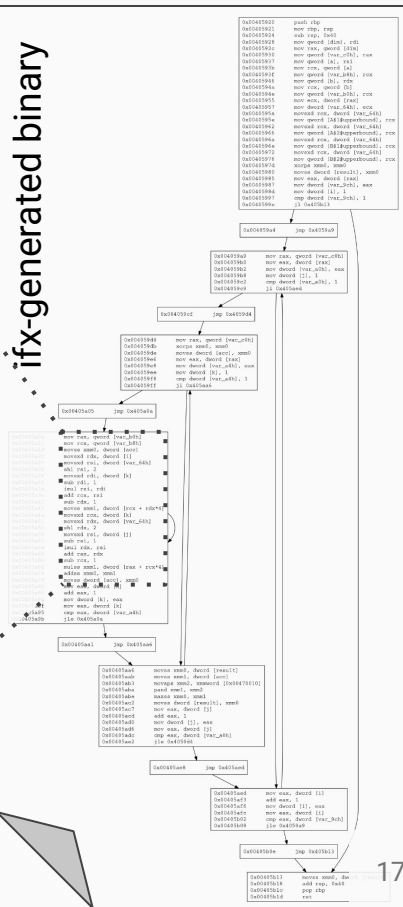
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Read Operands

```
0x7f9dea0970a4
```

ifx-generated binary





## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

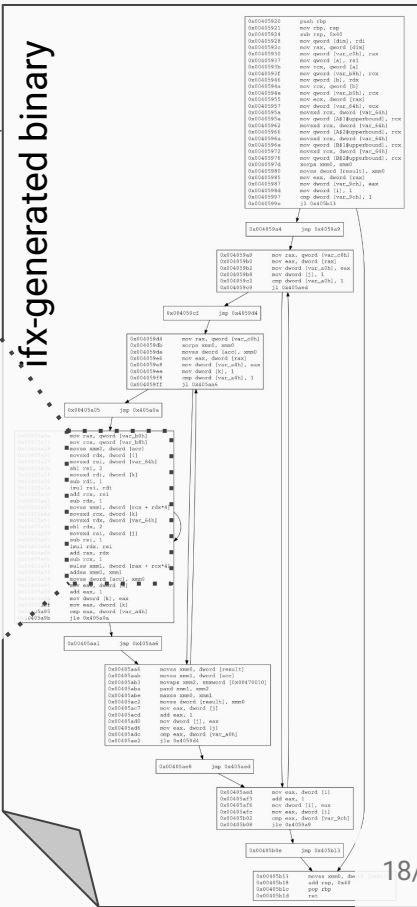
```

SMT  
query

## Active Symbols

```
xmm1
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

## ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT  
query

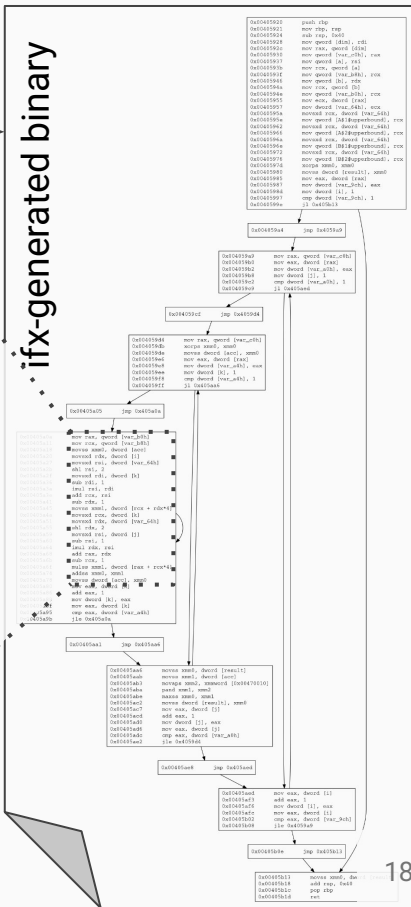
Active Symbols

```
xmm1
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

Read Operands

```
xmm1
0x7f9dea0970b4
```

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Active Symbols

xmm1

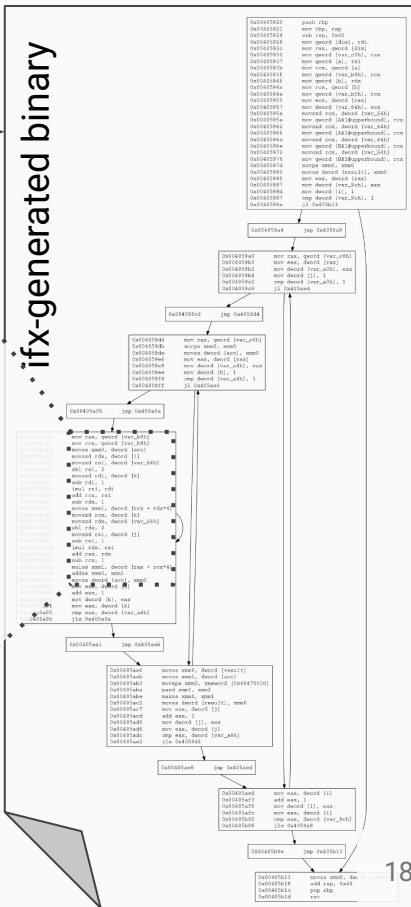
```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

Read Operands

xmm1

0x7f9dea0970b4

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm1_1  
    ( fp.mul rm  
        xmm1_0  
        0x7f9dea0970b4_0  
    )  
)
```

SMT  
query

Active Symbols

xmm1

0x7f9dea0970a4  
0x7f9dea0970a8  
0x7f9dea0970ac  
0x7f9dea0970b0  
**0x7f9dea0970b4**  
0x7f9dea0970b8  
0x7f9dea0970bc  
0x7f9dea0970c0

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

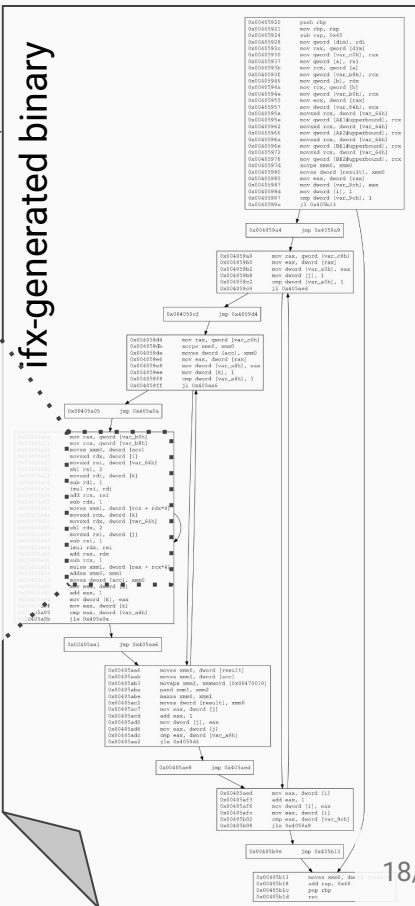
```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
... -> accumulate and save acc
```

Read Operands

xmm1

0x7f9dea0970b4

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =  
    xmm1_1  
    ( fp.mul rm  
        xmm1_0  
        0x7f9dea0970b4_0  
    )  
)
```

```
assert ( fp.isNaN xmm1_1 )
```

SMT  
query

Active Symbols

xmm1

0x7f9dea0970a4  
0x7f9dea0970a8  
0x7f9dea0970ac  
0x7f9dea0970b0  
**0x7f9dea0970b4**  
0x7f9dea0970b8  
0x7f9dea0970bc  
0x7f9dea0970c0

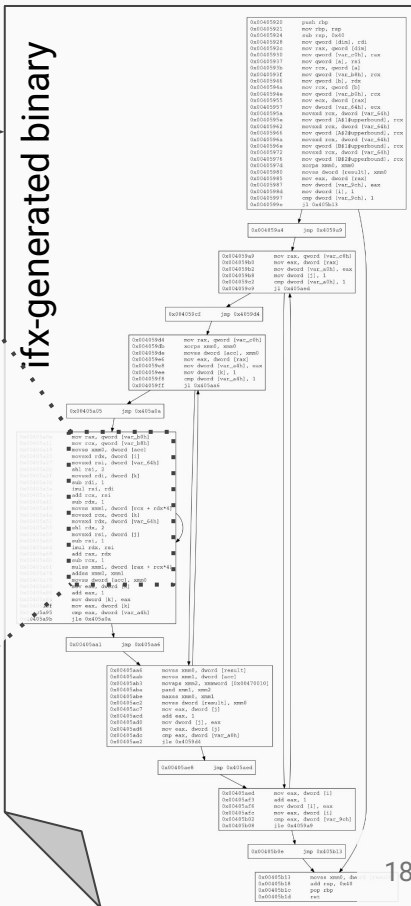
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
...  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
...  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
... -> accumulate and save acc
```

Read Operands

xmm1  
**0x7f9dea0970b4**

ifx-generated binary



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

```

## Read Operands

SMT  
query

## Active Symbols

xmm1

```
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

```
xmm1
xmm0
```

## ifx-generated binary

[illegible]

```

0x04859a4      jmp 0x059a9
039a3      mov eax, qword [var_0fh]
039a5      mov eax, dword [eax]
039a7      mov dword [var_0fh], eax
039a9      mov dword [i], 1
039ab      cmp dword [var_0fh], 1

```

```
0x0040099cf      jmp 0x004009d4
```

```

4      mov eax, dword [var_5bh]
5      xorps xmm0, xmm0
6      movss dword [acc], xmm0
7      mov eax, dword [eax]
8      mov dword [var_5bh], eax
9      mov dword [i], 1
a      cmp dword [var_5bh], 1
f      jle 0x004009aa5

```

```

0x00400500: mov rax, qword [
0x00400504: mov rcx, qword [
0x00400508: movss xmm0, dword
0x0040050C: movssd r1d, dword
0x00400510: movssd r1i, dword
0x00400514: shl rax, 2
0x00400518: movssd r1d, dword
0x0040051C: sub r1d, 1
0x00400520: jmp rax, rdi

```

```

0x02403441 mov rdx, 1
0x02403443 movs xmm1, dword
0x02403445 movsdi rdx, dword
0x02403447 movsdi rdx, dword
0x02403449 shl rdx, 2
0x0240344b movsdi rsi, dword
0x0240344d mov rsi, 1
0x0240344f leal rdx, rsi
0x02403451 add rax, rdx
0x02403453 mov rax, 1
0x02403455 movs xmm1, dword
0x02403457 addsd xmm1, xmm1
0x02403459 movs dword [acc
0x0240345b

```

```

0x00401000 mov dword [K], 0
0x00401001 mov eax, dword [K]
0x00401002 cmp eax, dword [K]
0x00401003 jle 0x00401004
0x00401004
0x00401005
0x00401006
0x00401007
0x00401008
0x00401009
0x0040100A
0x0040100B
0x0040100C
0x0040100D
0x0040100E
0x0040100F
0x00401010
0x00401011
0x00401012
0x00401013
0x00401014
0x00401015
0x00401016
0x00401017
0x00401018
0x00401019
0x0040101A
0x0040101B
0x0040101C
0x0040101D
0x0040101E
0x0040101F
0x00401020
0x00401021
0x00401022
0x00401023
0x00401024
0x00401025
0x00401026
0x00401027
0x00401028
0x00401029
0x0040102A
0x0040102B
0x0040102C
0x0040102D
0x0040102E
0x0040102F
0x00401030
0x00401031
0x00401032
0x00401033
0x00401034
0x00401035
0x00401036
0x00401037
0x00401038
0x00401039
0x0040103A
0x0040103B
0x0040103C
0x0040103D
0x0040103E
0x0040103F
0x00401040
0x00401041
0x00401042
0x00401043
0x00401044
0x00401045
0x00401046
0x00401047
0x00401048
0x00401049
0x0040104A
0x0040104B
0x0040104C
0x0040104D
0x0040104E
0x0040104F
0x00401050
0x00401051
0x00401052
0x00401053
0x00401054
0x00401055
0x00401056
0x00401057
0x00401058
0x00401059
0x0040105A
0x0040105B
0x0040105C
0x0040105D
0x0040105E
0x0040105F
0x00401060
0x00401061
0x00401062
0x00401063
0x00401064
0x00401065
0x00401066
0x00401067
0x00401068
0x00401069
0x0040106A
0x0040106B
0x0040106C
0x0040106D
0x0040106E
0x0040106F
0x00401070
0x00401071
0x00401072
0x00401073
0x00401074
0x00401075
0x00401076
0x00401077
0x00401078
0x00401079
0x0040107A
0x0040107B
0x0040107C
0x0040107D
0x0040107E
0x0040107F
0x00401080
0x00401081
0x00401082
0x00401083
0x00401084
0x00401085
0x00401086
0x00401087
0x00401088
0x00401089
0x0040108A
0x0040108B
0x0040108C
0x0040108D
0x0040108E
0x0040108F
0x00401090
0x00401091
0x00401092
0x00401093
0x00401094
0x00401095
0x00401096
0x00401097
0x00401098
0x00401099
0x0040109A
0x0040109B
0x0040109C
0x0040109D
0x0040109E
0x0040109F
0x004010A0
0x004010A1
0x004010A2
0x004010A3
0x004010A4
0x004010A5
0x004010A6
0x004010A7
0x004010A8
0x004010A9
0x004010AA
0x004010AB
0x004010AC
0x004010AD
0x004010AE
0x004010AF
0x004010B0
0x004010B1
0x004010B2
0x004010B3
0x004010B4
0x004010B5
0x004010B6
0x004010B7
0x004010B8
0x004010B9
0x004010BA
0x004010BB
0x004010BC
0x004010BD
0x004010BE
0x004010BF
0x004010C0
0x004010C1
0x004010C2
0x004010C3
0x004010C4
0x004010C5
0x004010C6
0x004010C7
0x004010C8
0x004010C9
0x004010CA
0x004010CB
0x004010CC
0x004010CD
0x004010CE
0x004010CF
0x004010D0
0x004010D1
0x004010D2
0x004010D3
0x004010D4
0x004010D5
0x004010D6
0x004010D7
0x004010D8
0x004010D9
0x004010DA
0x004010DB
0x004010DC
0x004010DD
0x004010DE
0x004010DF
0x004010E0
0x004010E1
0x004010E2
0x004010E3
0x004010E4
0x004010E5
0x004010E6
0x004010E7
0x004010E8
0x004010E9
0x004010EA
0x004010EB
0x004010EC
0x004010ED
0x004010EE
0x004010EF
0x004010F0
0x004010F1
0x004010F2
0x004010F3
0x004010F4
0x004010F5
0x004010F6
0x004010F7
0x004010F8
0x004010F9
0x004010FA
0x004010FB
0x004010FC
0x004010FD
0x004010FE
0x004010FF
0x00401100
0x00401101
0x00401102
0x00401103
0x00401104
0x00401105
0x00401106
0x00401107
0x00401108
0x00401109
0x0040110A
0x0040110B
0x0040110C
0x0040110D
0x0040110E
0x0040110F
0x00401110
0x00401111
0x00401112
0x00401113
0x00401114
0x00401115
0x00401116
0x00401117
0x00401118
0x00401119
0x0040111A
0x0040111B
0x0040111C
0x0040111D
0x0040111E
0x0040111F
0x00401120
0x00401121
0x00401122
0x00401123
0x00401124
0x00401125
0x00401126
0x00401127
0x00401128
0x00401129
0x0040112A
0x0040112B
0x0040112C
0x0040112D
0x0040112E
0x0040112F
0x00401130
0x00401131
0x00401132
0x00401133
0x00401134
0x00401135
0x00401136
0x00401137
0x00401138
0x00401139
0x0040113A
0x0040113B
0x0040113C
0x0040113D
0x0040113E
0x0040113F
0x00401140
0x00401141
0x00401142
0x00401143
0x00401144
0x00401145
0x00401146
0x00401147
0x00401148
0x00401149
0x0040114A
0x0040114B
0x0040114C
0x0040114D
0x0040114E
0x0040114F
0x00401150
0x00401151
0x00401152
0x00401153
0x00401154
0x00401155
0x00401156
0x00401157
0x00401158
0x00401159
0x0040115A
0x0040115B
0x0040115C
0x0040115D
0x0040115E
0x0040115F
0x00401160
0x00401161
0x00401162
0x00401163
0x00401164
0x00401165
0x00401166
0x00401167
0x00401168
0x00401169
0x0040116A
0x0040116B
0x0040116C
0x0040116D
0x0040116E
0x0040116F
0x00401170
0x00401171
0x00401172
0x00401173
0x0
```

```

movzx eax, xmm0
movss dword [result], xmm0
mov eax, dword [i]
add eax, 1
mov dword [i], eax
mov eax, dword [i]
cmp eax, dword [var_48h]
jle 0x4059d4

```

0x4040e8 jmp 0x4059d4

|            |   |
|------------|---|
| 0x00483a00 | 0 |
| 0x00483a01 | 0 |
| 0x00483a02 | 0 |
| 0x00483a03 | 0 |
| 0x00483a04 | 0 |
| 0x00483a05 | 0 |
| 0x00483a06 | 0 |
| 0x00483a07 | 0 |
| 0x00483a08 | 0 |
| 0x00483a09 | 0 |
| 0x00483a0a | 0 |
| 0x00483a0b | 0 |
| 0x00483a0c | 0 |
| 0x00483a0d | 0 |
| 0x00483a0e | 0 |
| 0x00483a0f | 0 |
| 0x00483a10 | 0 |
| 0x00483a11 | 0 |
| 0x00483a12 | 0 |
| 0x00483a13 | 0 |
| 0x00483a14 | 0 |
| 0x00483a15 | 0 |
| 0x00483a16 | 0 |
| 0x00483a17 | 0 |
| 0x00483a18 | 0 |
| 0x00483a19 | 0 |
| 0x00483a1a | 0 |
| 0x00483a1b | 0 |
| 0x00483a1c | 0 |
| 0x00483a1d | 0 |
| 0x00483a1e | 0 |
| 0x00483a1f | 0 |
| 0x00483a20 | 0 |
| 0x00483a21 | 0 |
| 0x00483a22 | 0 |
| 0x00483a23 | 0 |
| 0x00483a24 | 0 |
| 0x00483a25 | 0 |
| 0x00483a26 | 0 |
| 0x00483a27 | 0 |
| 0x00483a28 | 0 |
| 0x00483a29 | 0 |
| 0x00483a2a | 0 |
| 0x00483a2b | 0 |
| 0x00483a2c | 0 |
| 0x00483a2d | 0 |
| 0x00483a2e | 0 |
| 0x00483a2f | 0 |
| 0x00483a30 | 0 |
| 0x00483a31 | 0 |
| 0x00483a32 | 0 |
| 0x00483a33 | 0 |
| 0x00483a34 | 0 |
| 0x00483a35 | 0 |
| 0x00483a36 | 0 |
| 0x00483a37 | 0 |
| 0x00483a38 | 0 |
| 0x00483a39 | 0 |
| 0x00483a3a | 0 |
| 0x00483a3b | 0 |
| 0x00483a3c | 0 |
| 0x00483a3d | 0 |
| 0x00483a3e | 0 |
| 0x00483a3f | 0 |
| 0x00483a40 | 0 |
| 0x00483a41 | 0 |
| 0x00483a42 | 0 |
| 0x00483a43 | 0 |
| 0x00483a44 | 0 |
| 0x00483a45 | 0 |
| 0x00483a46 | 0 |
| 0x00483a47 | 0 |
| 0x00483a48 | 0 |
| 0x00483a49 | 0 |
| 0x00483a4a | 0 |
| 0x00483a4b | 0 |
| 0x00483a4c | 0 |
| 0x00483a4d | 0 |
| 0x00483a4e | 0 |
| 0x00483a4f | 0 |
| 0x00483a50 | 0 |
| 0x00483a51 | 0 |
| 0x00483a52 | 0 |
| 0x00483a53 | 0 |
| 0x00483a54 | 0 |
| 0x00483a55 | 0 |
| 0x00483a56 | 0 |
| 0x00483a57 | 0 |
| 0x00483a58 | 0 |
| 0x00483a59 | 0 |
| 0x00483a5a | 0 |
| 0x00483a5b | 0 |
| 0x00483a5c | 0 |
| 0x00483a5d | 0 |
| 0x00483a5e | 0 |
| 0x00483a5f | 0 |
| 0x00483a60 | 0 |
| 0x00483a61 | 0 |
| 0x00483a62 | 0 |
| 0x00483a63 | 0 |
| 0x00483a64 | 0 |
| 0x00483a65 | 0 |
| 0x00483a66 | 0 |
| 0x00483a67 | 0 |
| 0x00483a68 | 0 |
| 0x00483a69 | 0 |
| 0x00483a6a | 0 |
| 0x00483a6b | 0 |
| 0x00483a6c | 0 |
| 0x00483a6d | 0 |
| 0x00483a6e | 0 |
| 0x00483a6f | 0 |
| 0x00483a70 | 0 |
| 0x00483a71 | 0 |
| 0x00483a72 | 0 |
| 0x00483a73 | 0 |
| 0x00483a74 | 0 |
| 0x00483a75 | 0 |
| 0x00483a76 | 0 |
| 0x00483a77 | 0 |
| 0x00483a78 | 0 |
| 0x00483a79 | 0 |
| 0x00483a7a | 0 |
| 0x00483a7b | 0 |
| 0x00483a7c | 0 |
| 0x00483a7d | 0 |
| 0x00483a7e | 0 |
| 0x00483a7f | 0 |
| 0x00483a80 | 0 |
| 0x00483a81 | 0 |
| 0x00483a82 | 0 |
| 0x00483a83 | 0 |
| 0x00483a84 | 0 |
| 0x00483a85 | 0 |
| 0x00483a86 | 0 |
| 0x00483a87 | 0 |
| 0x         |   |

|           |         |
|-----------|---------|
| 0x0405010 | pop rbp |
| 0x0405014 | ret     |

**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( =
    xmm0_0
    ( fp.add rm
        #b00000000000000000000000000000000
        xmm1_1
    )
)
```

SMT  
query

## Active Symbols

xmm1

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

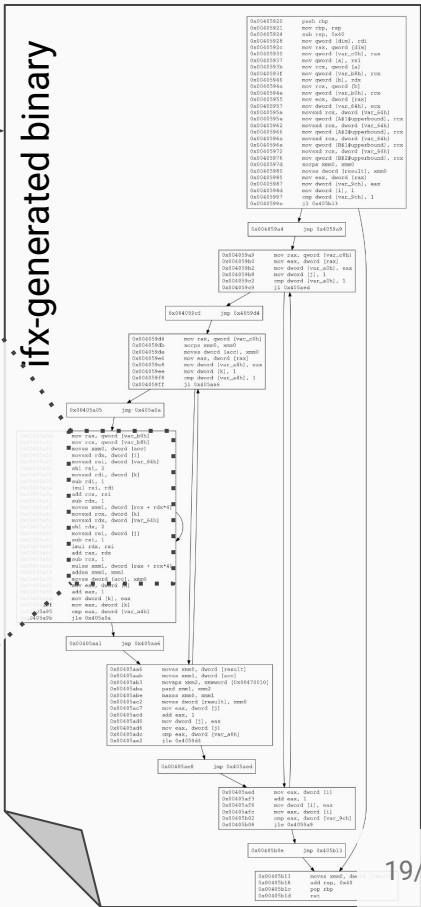
```

## Read Operands

xmm1  
xmm0

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

## ifx-generated binary



**Approach: encode the desired behavior (control flow + exception) into an SMT query**

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```

assert ( =
    xmm0_0 }
    ( fp.add rm
      #b00000000000000000000000000000000
      xmm1_1
    )
)

```

SMT  
query

# Active Symbols

```

0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0

```

```

:      -> load acc
movss xmm0, dword ptr [rbp-0x98]

:      -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]

:      -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
:      -> accumulate and save acc

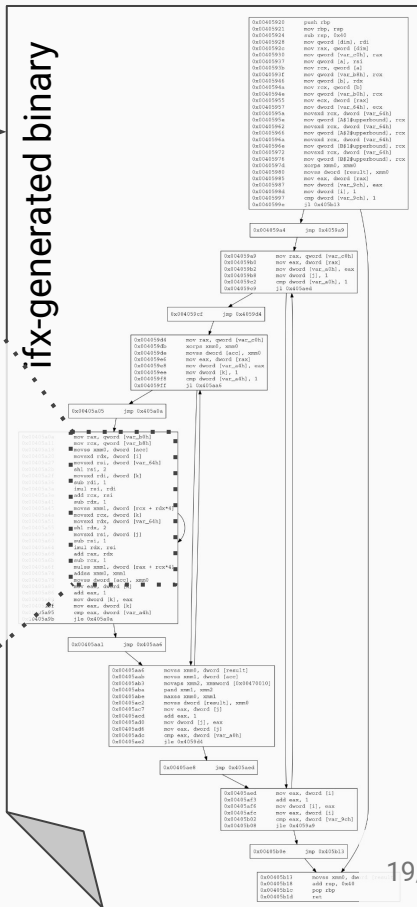
```

## Read Operands

```
xmm1
xmm0
```

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

ifx-generated binary





# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

SMT  
query

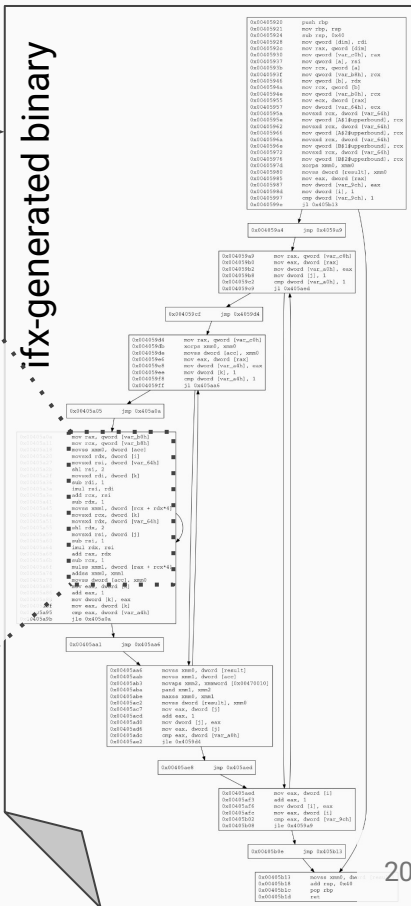
Active Symbols

```
xmm0
xmm1
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

Read Operands

xmm0

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = 0x7ffe9945e28_1 xmm0_0 )
```

SMT  
query

Active Symbols

```
xmm0
xmm1
0x7f9dea0970a4
0x7f9dea0970a8
0x7f9dea0970ac
0x7f9dea0970b0
0x7f9dea0970b4
0x7f9dea0970b8
0x7f9dea0970bc
0x7f9dea0970c0
```

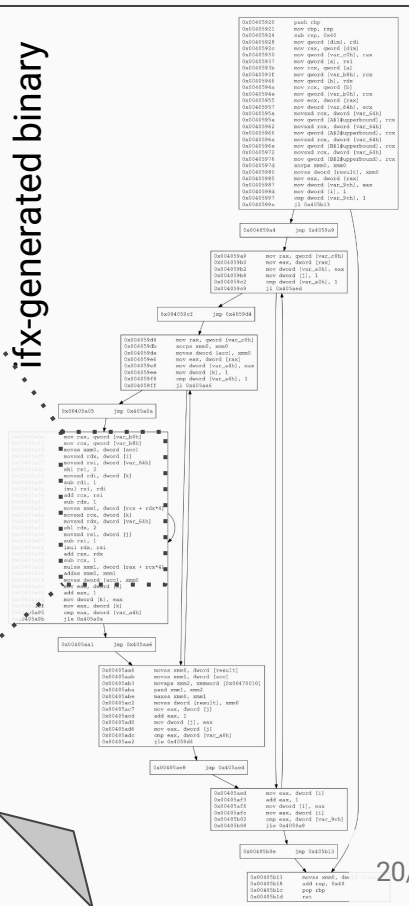
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc
movss xmm0, dword ptr [rbp-0x98]
...
... -> load A(i,k)
movss xmm1, dword ptr [rcx+rdx*4]
...
... -> multiply with B(k,j)
mulss xmm1, dword ptr [rax+rcx*4]
addss xmm0, xmm1
movss dword ptr [rbp-0x98], xmm0
... -> accumulate and save acc
```

Read Operands

xmm0

ifx-generated binary



# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

```
assert ( = 0x7ffef9945e28_1 xmm0_0 )
```

SMT  
query

Active Symbols

0x7ffef9945e28

xmm0

xmm1

0x7f9dea0970a4  
0x7f9dea0970a8  
0x7f9dea0970ac  
0x7f9dea0970b0  
0x7f9dea0970b4  
0x7f9dea0970b8  
0x7f9dea0970bc  
0x7f9dea0970c0

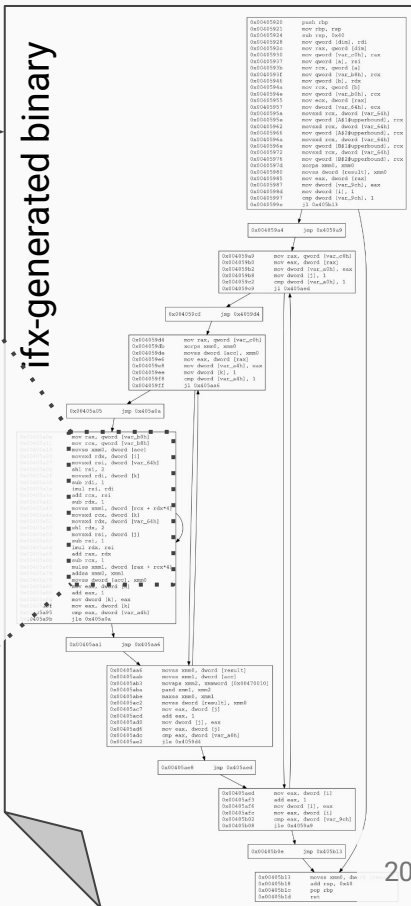
$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$B = \begin{bmatrix} 4 & 2 \\ 3 & 1 \end{bmatrix}$$

```
... -> load acc  
movss xmm0, dword ptr [rbp-0x98]  
... -> load A(i,k)  
movss xmm1, dword ptr [rcx+rdx*4]  
... -> multiply with B(k,j)  
mulss xmm1, dword ptr [rax+rcx*4]  
addss xmm0, xmm1  
movss dword ptr [rbp-0x98], xmm0  
... -> accumulate and save acc
```

Read Operands

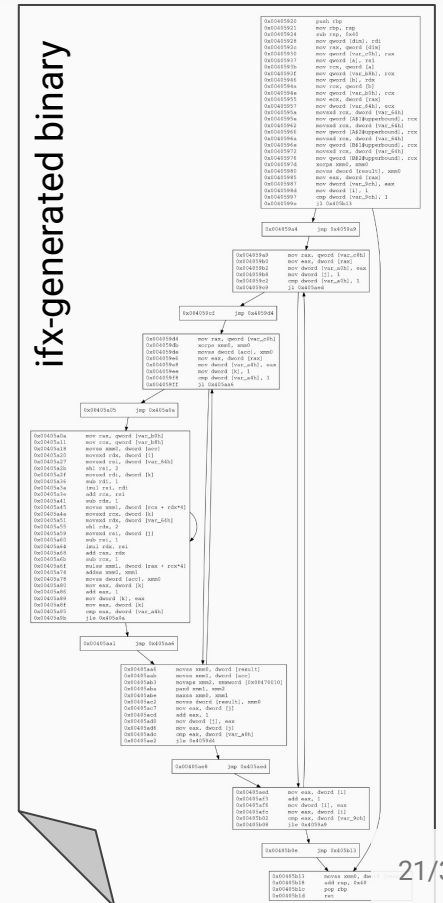
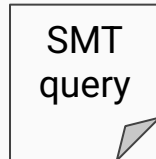
xmm0

ifx-generated binary



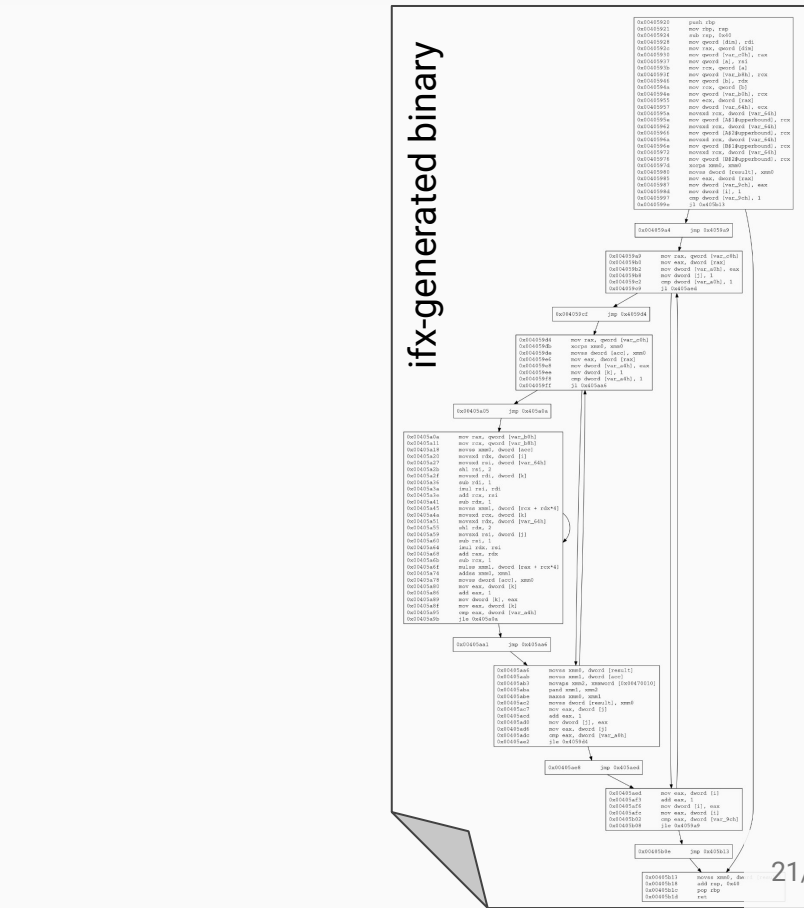
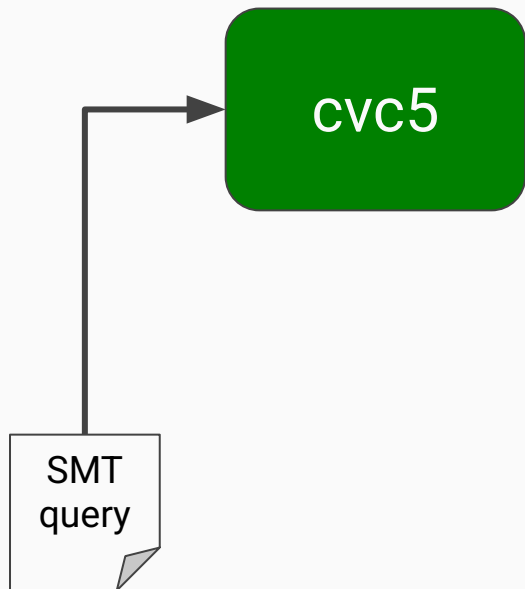
## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.



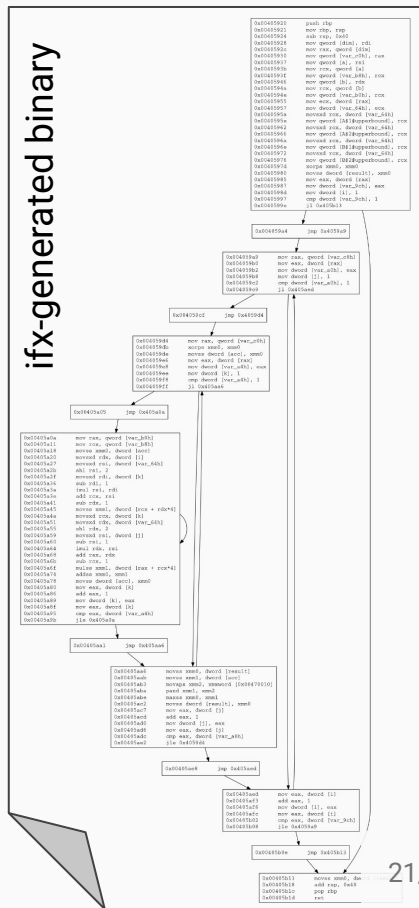
# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} NaN & 0 \\ 0 & 0 \end{bmatrix}$$
$$B = \begin{bmatrix} -1.17 \times 10^{-38} & 0 \\ 0 & 0 \end{bmatrix}$$

cvc5

SMT  
query



## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

$$A = \begin{bmatrix} NaN & 0 \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} -1.17 \times 10^{-38} & 0 \\ 0 & 0 \end{bmatrix}$$

# cvc5

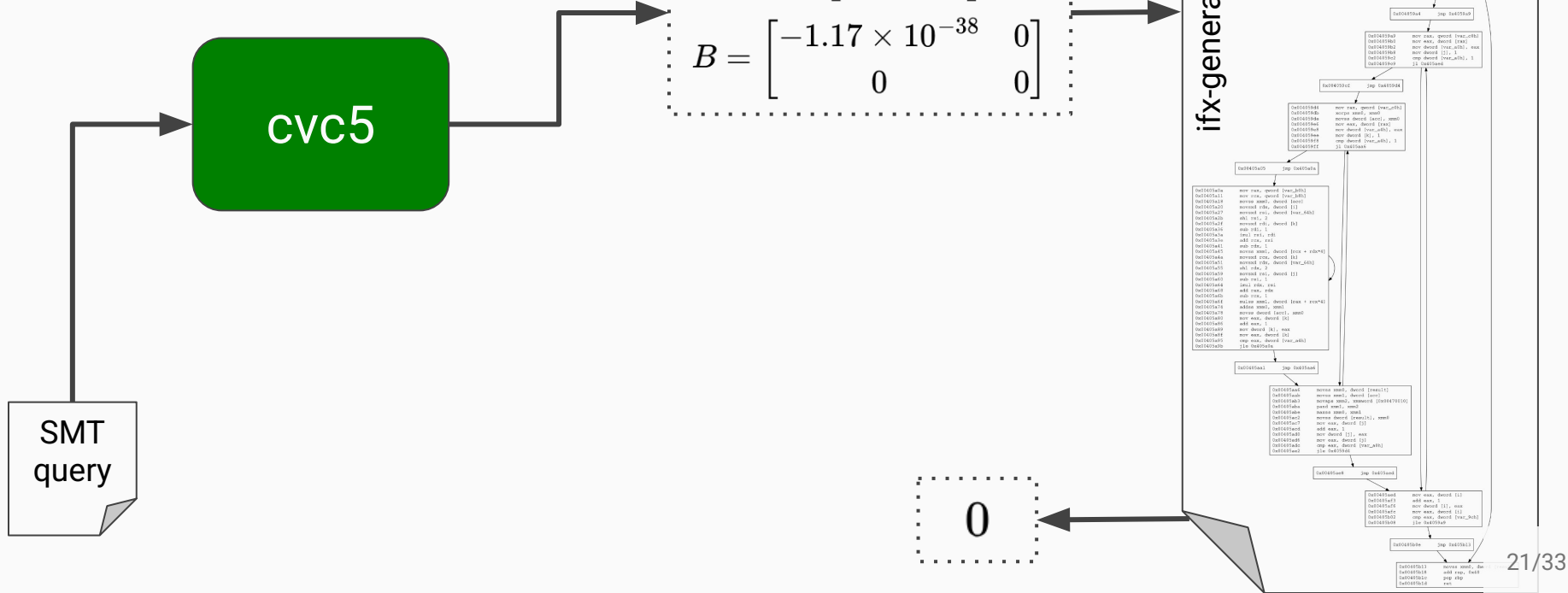
SMT  
query

## ifx-generated binary

21/33

## Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.





# Approach: encode the desired behavior (control flow + exception) into an SMT query

Let's check the *first warning* which resulted from spoofing an Invalid exception in the first execution of the multiply instruction.

cvc5

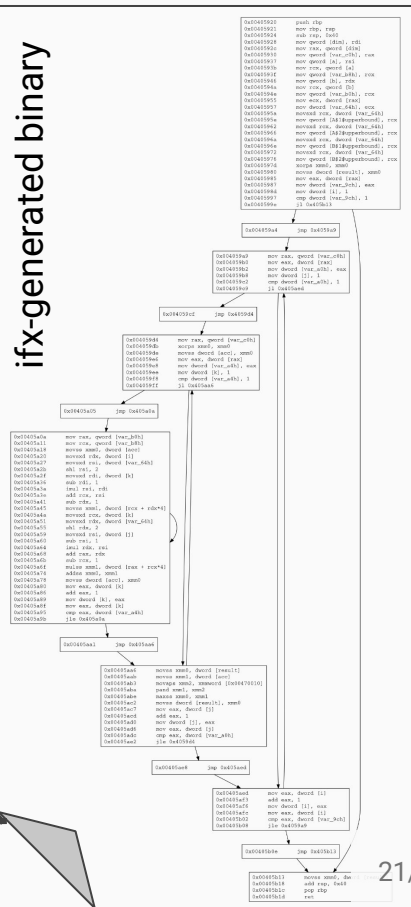
$$A = \begin{bmatrix} NaN & 0 \\ 0 & 0 \end{bmatrix}$$
$$B = \begin{bmatrix} -1.17 \times 10^{-38} & 0 \\ 0 & 0 \end{bmatrix}$$

| Disassembly                       | Source Location | Event | Taint Count |
|-----------------------------------|-----------------|-------|-------------|
| movss xmm1, dword ptr [rcx+rdx*4] | main.f90:17     | G---  | 1           |
| mulss xmm1, dword ptr [rax+rcx*4] | main.f90:17     | -P-r  | 1           |
| addss xmm0, xmm1                  | main.f90:17     | -P-r  | 2           |
| ...                               |                 |       |             |
| maxss xmm0, xmm1                  | main.f90:19     | --Kr  | 1           |
| movss dword ptr [rbp-0x44], xmm0  | main.f90:19     | --K-  | 0           |

SMT  
query

0

ifx-generated binary

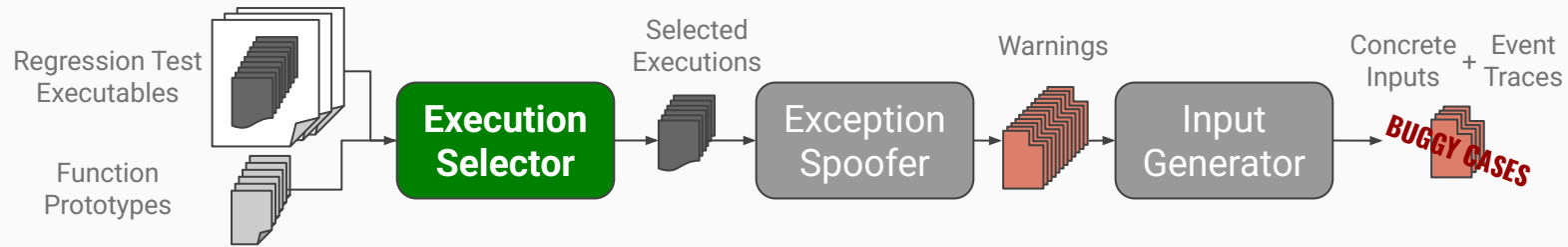


# Challenge 3: How do we get a representative set of function executions to test?

Soundness increases as the set of function executions covers more possible execution paths...

**...but covering all possible execution paths is generally not feasible.**

# Approach: Take executions from the library's regression tests



# Evaluation: A BLAS Case Study

- 26 functions from Levels 1 & 2 of the BLAS
  - Implementations taken from the **Reference BLAS**, **OpenBLAS**, and **BLIS**
  - Binaries generated by **GNU** (gfortran, gcc) and **Intel** (ifx, icx) compilers
    - different combinations of default, -O3, and -ffast-math/-fp-model=fast=[1|2] optimizations
- > 598 (function, implementation, compiler, optimizations) tuples
- > 12 hours of total testing time

## Finding 1: Only 4.4% of spoofed exceptions resulted in warnings

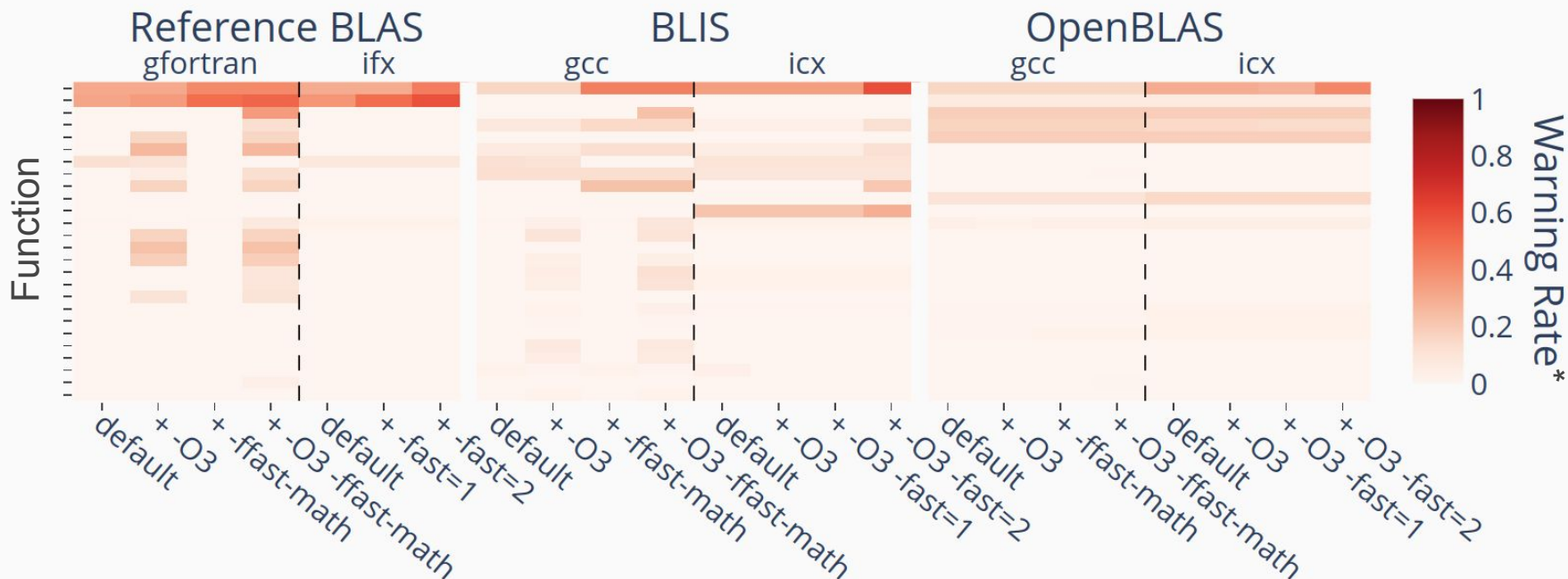
Out of 530K spoofed exceptions, 23K resulted in warnings

-> supports the assumption that most code handles exceptions correctly

# Finding 1: Only 4.4% of spoofed exceptions resulted in warnings

Out of 530K spoofed exceptions, 23K resulted in warnings

-> supports the assumption that most code handles exceptions correctly



\*Warning Rate = ( # exception-handling failures / # spoofed exceptions ) 25/33

## Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

# Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

**We find three main causes for the failures:**



# Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

**We find three main causes for the failures:**

- 1) Compiler optimizations changing control flow

# Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

**We find three main causes for the failures:**

- 1) Compiler optimizations changing control flow
- 2) Design/documentation not accounting for NaNs or Infs

# Finding 2: EXCVATE found inputs triggering exception-handling failures in 5/26 BLAS functions

**We find three main causes for the failures:**

- 1) Compiler optimizations changing control flow
- 2) Design/documentation not accounting for NaNs or Infs
- 3) Implicit zeroes in input matrices

**sgemv / sger**

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found  
exception-handling failures  
caused by **compiler  
optimizations that change  
control flow** in the face of  
comparisons involving NaN

## sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found  
exception-handling failures  
caused by **compiler  
optimizations that change  
control flow** in the face of  
comparisons involving NaN

```
(in) TRANS: N  
(in) M: 1  
(in) N: 3  
(in) ALPHA: 1.14752e-41  
(in) LDA: 2  
(in) INCX: 1  
(in) BETA: 1  
(in) INCY: 1  
(in) A: 0 0 0 0 0 0  
(in) X: nan 0 0  
(in) Y: 0
```

# sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found  
exception-handling failures  
caused by **compiler  
optimizations that change  
control flow** in the face of  
comparisons involving NaN

```
(in) TRANS: N
(in) M: 1
(in) N: 3
(in) ALPHA: 1.14752e-41
(in) LDA: 2
(in) INCX: 1
(in) BETA: 1
(in) INCY: 1
(in) A: 0 0 0 0 0 0
(in) X: nan 0 0
(in) Y: 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found  
exception-handling failures  
caused by **compiler  
optimizations that change  
control flow** in the face of  
comparisons involving NaN

```
(in) TRANS: N  
(in) M: 1  
(in) N: 3  
(in) ALPHA: 1.14752e-41  
(in) LDA: 2  
(in) INCX: 1  
(in) BETA: 1  
(in) INCY: 1  
(in) A: 0 0 0 0 0 0  
(in) X: nan 0 0  
(in) Y: 0
```

|                         | Reference BLAS | BLIS       | OpenBLAS |
|-------------------------|----------------|------------|----------|
| <b>GNU default</b>      |                |            |          |
| w/ -O3                  |                |            |          |
| w/ -ffast-math          |                | (out) Y: 0 |          |
| w/ -O3 -ffast-math      |                | (out) Y: 0 |          |
| <b>Intel default</b>    |                |            |          |
| w/ -O3                  | (out) Y: nan   |            |          |
| w/ -O3 -fp-model=fast=1 |                |            |          |
| w/ -O3 -fp-model=fast=2 |                | (out) Y: 0 |          |

# sgemv / sger

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

$$\mathbf{A} := \alpha \mathbf{x} \mathbf{y}^T + \mathbf{A}$$

EXCVATE found  
exception-handling failures  
caused by **compiler  
optimizations that change  
control flow** in the face of  
comparisons involving NaN

```
(in) M: 2  
(in) N: 1  
(in) ALPHA: 1.14752e-41  
(in) INCX: 1  
(in) INCY: 1  
(in) LDA: 3  
(in) X: 0 0  
(in) Y: nan  
(in) A: 0 0 0
```

|                         | Reference BLAS | BLIS           | OpenBLAS           |
|-------------------------|----------------|----------------|--------------------|
| <b>GNU default</b>      |                |                |                    |
| w/ -O3                  |                |                |                    |
| w/ -ffast-math          | (out) A: 0 0 0 | (out) A: 0 0 0 |                    |
| w/ -O3 -ffast-math      | (out) A: 0 0 0 | (out) A: 0 0 0 |                    |
| <b>Intel default</b>    |                |                |                    |
| w/ -O3                  |                |                | (out) A: nan nan 0 |
| w/ -O3 -fp-model=fast=1 |                |                |                    |
| w/ -O3 -fp-model=fast=2 | (out) A: 0 0 0 | (out) A: 0 0 0 |                    |



# srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# srotmg / srotm

Generate and apply a  
modified Givens rotation,  
respectively

EXCVATE found multiple  
different exception-handling  
failures **necessitating clearer  
documentation and even  
possible deprecation**

```
(in) N: 1  
(in) INCX: 1  
(in) INCY: 1  
(in) SPARAM: nan 0 0 0 0  
(in) SX: 0  
(in) SY: 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# srotmg / srotm

Generate and apply a  
modified Givens rotation,  
respectively

EXCVATE found multiple  
different exception-handling  
failures **necessitating clearer  
documentation and even  
possible deprecation**

```
(in) N: 1  
(in) INCX: 1  
(in) INCY: 1  
(in) SPARAM: nan 0 0 0 0  
(in) SX: 0  
(in) SY: 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

```
(out) SX: 0  
(out) SY: 0
```

# srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# srotmg / srotm

Generate and apply a  
modified Givens rotation,  
respectively

EXCVATE found multiple  
different exception-handling  
failures **necessitating clearer  
documentation and even  
possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

```
(out) SD1: 0
(out) SD2: 0
(out) SX1: 0
(out) SPARAM: -1 0 0 0 0
```

# srotmg / srotm

Generate and apply a  
modified Givens rotation,  
respectively

EXCVATE found multiple  
different exception-handling  
failures **necessitating clearer  
documentation and even  
possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.99976
(in) SX1: nan
(in) SY1: -inf
(in) SPARAM: 0 0 0 0 0
```

```
(in) SD1: 1.17549e-38
(in) SD2: -1.4013e-45
(in) SX1: nan
(in) SY1: -1.66667
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| GNU default             |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| Intel default           |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

(out) SD1: 0  
(out) SD2: 0  
(out) SX1: 0  
(out) SPARAM: -1 0 0 0 0

**UNDOCUMENTED  
ERROR CODE**

# srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS                    | OpenBLAS       |
|-------------------------|----------------|-------------------------|----------------|
| <b>GNU default</b>      |                |                         |                |
| w/ -O3                  |                |                         |                |
| w/ -ffast-math          |                |                         |                |
| w/ -O3 -ffast-math      |                |                         |                |
| <b>Intel default</b>    |                |                         |                |
| w/ -O3                  |                |                         |                |
| w/ -O3 -fp-model=fast=1 |                |                         |                |
| w/ -O3 -fp-model=fast=2 |                | Undocumented Error Code | NaNs in output |



# srotmg / srotm

Generate and apply a modified Givens rotation, respectively

EXCVATE found multiple different exception-handling failures **necessitating clearer documentation and even possible deprecation**

```
(in) SD1: inf
(in) SD2: -1.23382e-05
(in) SX1: -1.81899e-12
(in) SY1: -2.21834e-39
(in) SPARAM: 0 0 0 0 0
```

|                         | Reference BLAS | BLIS                    | OpenBLAS       |
|-------------------------|----------------|-------------------------|----------------|
| <b>GNU default</b>      |                |                         |                |
| w/ -O3                  |                |                         |                |
| w/ -ffast-math          |                |                         |                |
| w/ -O3 -ffast-math      |                |                         |                |
| <b>Intel default</b>    |                |                         |                |
| w/ -O3                  |                |                         |                |
| w/ -O3 -fp-model=fast=1 |                |                         |                |
| w/ -O3 -fp-model=fast=2 |                | Undocumented Error Code | NaNs in output |

**BLAS TIMEOUT**

# sgbmv

$$\mathbf{y} := \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation**.

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# sgbmv

$$y := \alpha Ax + \beta y$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation.**

```
(in) TRANS: N
(in) M: 2
(in) N: 5
(in) KL: 0
(in) KU: 0
(in) ALPHA: 1
(in) LDA: 2
(in) INCX: -1
(in) BETA: 0
(in) INCY: 1
(in) A: 4.26326e-14 -1 -1
-1 -1 -1 -1 -1 -1 -1
(in) X: nan -1 -1 -1 0
(in) Y: 0 -1
```

$$\begin{bmatrix} 4.26326 \times 10^{-14} & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \\ NaN \end{bmatrix}$$

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

# sgbmv

$$y := \alpha Ax + \beta y$$

EXCVATE found exception handling failures in all tuples due to the handling of **implicit zeros in the banded matrix representation.**

```
(in) TRANS: N
(in) M: 2
(in) N: 5
(in) KL: 0
(in) KU: 0
(in) ALPHA: 1
(in) LDA: 2
(in) INCX: -1
(in) BETA: 0
(in) INCY: 1
(in) A: 4.26326e-14 -1 -1
-1 -1 -1 -1 -1 -1 -1
(in) X: nan -1 -1 -1 0
(in) Y: 0 -1
```

$$\begin{bmatrix} 4.26326 \times 10^{-14} & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ -1 \\ -1 \\ -1 \\ NaN \end{bmatrix}$$

|                         | Reference BLAS | BLIS | OpenBLAS |
|-------------------------|----------------|------|----------|
| <b>GNU default</b>      |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -ffast-math          |                |      |          |
| w/ -O3 -ffast-math      |                |      |          |
| <b>Intel default</b>    |                |      |          |
| w/ -O3                  |                |      |          |
| w/ -O3 -fp-model=fast=1 |                |      |          |
| w/ -O3 -fp-model=fast=2 |                |      |          |

(out) Y: 0 1

# We have...

## ...introduced the problem of testing exception handling

- And why current input-generation tools are a poor fit for the problem

## ...described a novel approach to this problem

- Targeting binary executables using exception spoofing and constraint solving
- Implemented in the prototype tool EXCVATE

## ...demonstrated our approach on the BLAS

- Tested across multiple implementations, compilers, compiler optimizations
- Found exception-handling failures in 5/26 functions

Source code and data available at <https://github.com/ucd-plse/EXCVATE>

# We have...

## ...introduced the problem of testing exception handling

- And why current input-generation tools are a poor fit for the problem

## ...described a novel approach to this problem

- Targeting binary executables using exception spoofing and constraint solving
- Implemented in the prototype tool EXCVATE

## ...demonstrated our approach on the BLAS

- Tested across multiple implementations, compilers, compiler optimizations
- Found exception-handling failures in 5/26 functions

Source code and data available at <https://github.com/ucd-plse/EXCVATE>

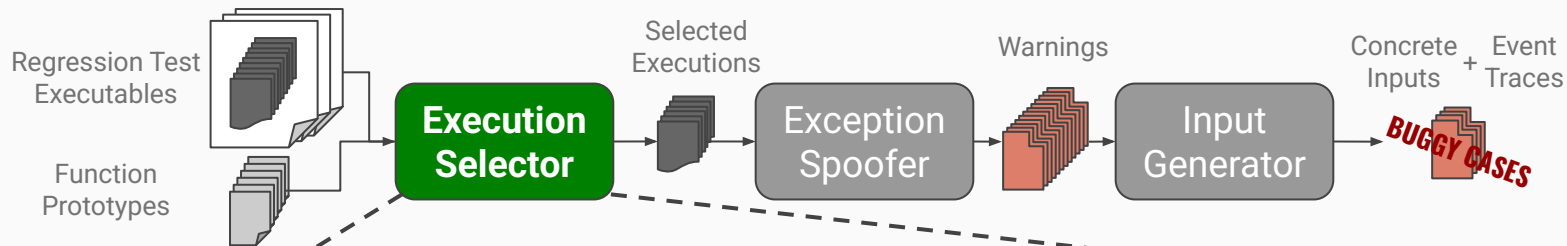
I am...

...looking for a job!!



# Appendices

# Approach: Take executions from the library's regression tests



For each function execution:

- ─ If we do not have a matching prototype:  
Skip

- ─ Construct ID for the function execution

- ─ If we have not already seen this ID:  
Save the function execution



# Future Work

- **Improve scalability**
  - Reducing redundant spoofs
  - Reducing redundant SMT queries
- **Improve soundness**
  - Increasing path coverage
  - Explore static methods
- **Support new targets**
  - Multithreaded programs
  - Complex-valued functions
  - More ISAs: FMA, AVX-512, PTX, CDNA