



Adventures in Fuzzing Instruction Selection



Overview

- Hardening instruction selection using fuzzers
- Motivated by Global ISel
- Leveraging libFuzzer to find backend bugs
- Techniques applicable to other parts of LLVM



Fuzzing Recap

- Using random inputs to find bugs
- Input generation
- Mutations of representative inputs
- Guided evolutionary fuzzing (afl-fuzz, libFuzzer)

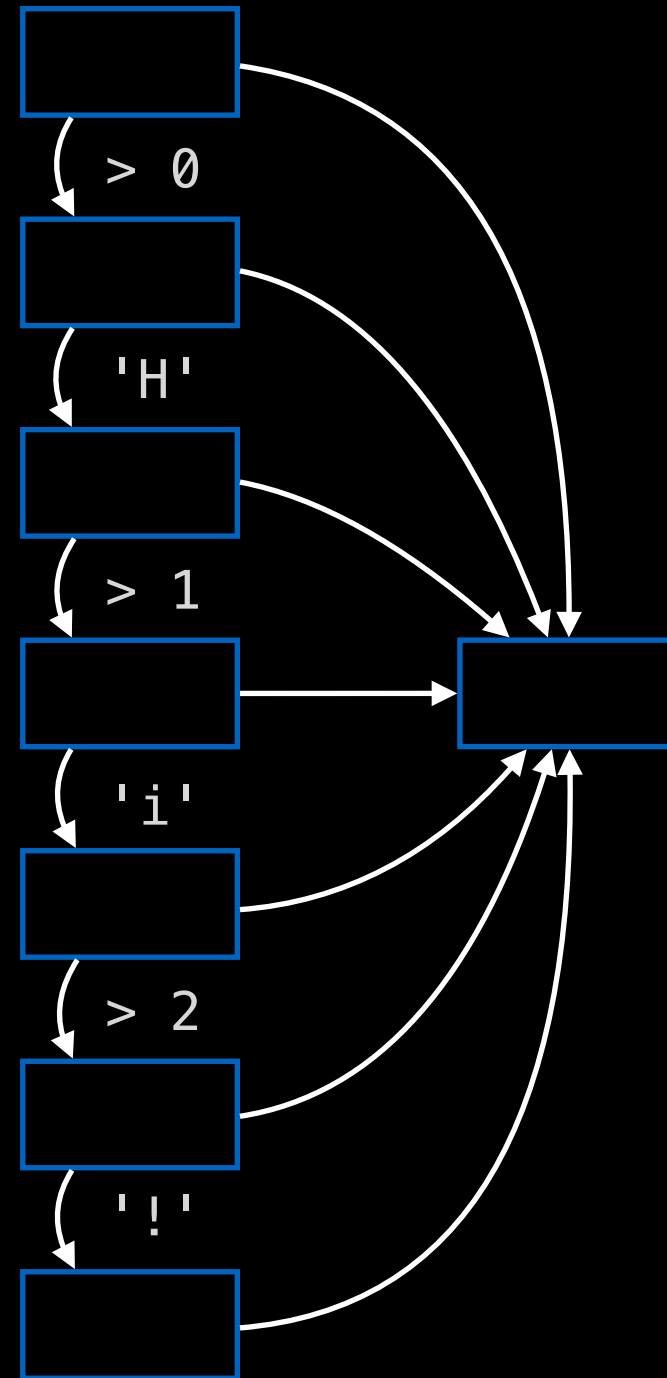


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
    const uint8_t *Data, size_t Size) {  
    if (Size > 0 && Data[0] == 'H')  
        if (Size > 1 && Data[1] == 'i')  
            if (Size > 2 && Data[2] == '!')  
                exit(0);  
    return 0;  
}
```

libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```



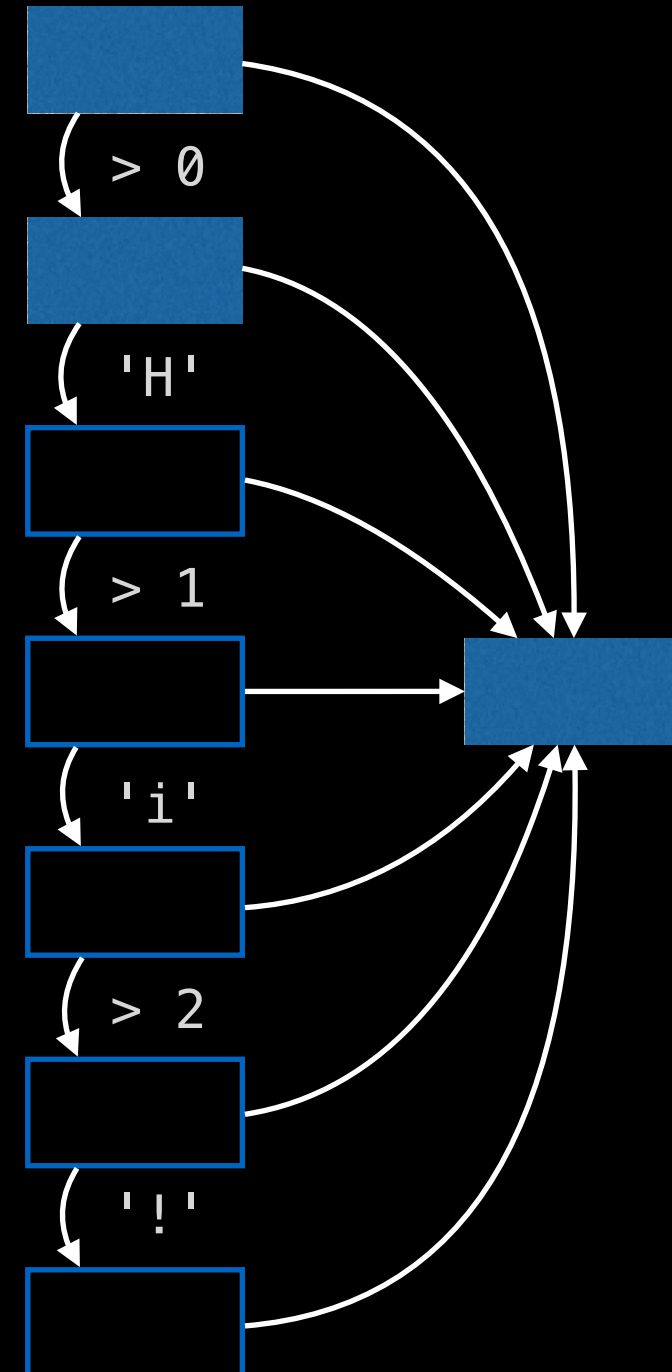
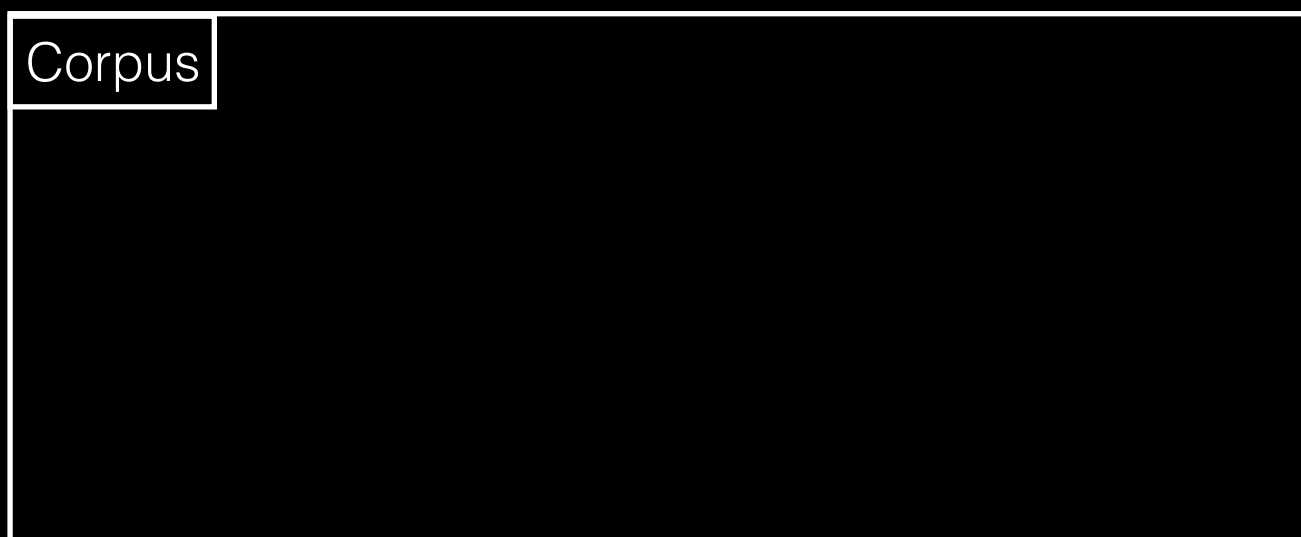
Corpus

libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: `<empty>`

Mutations: `q`

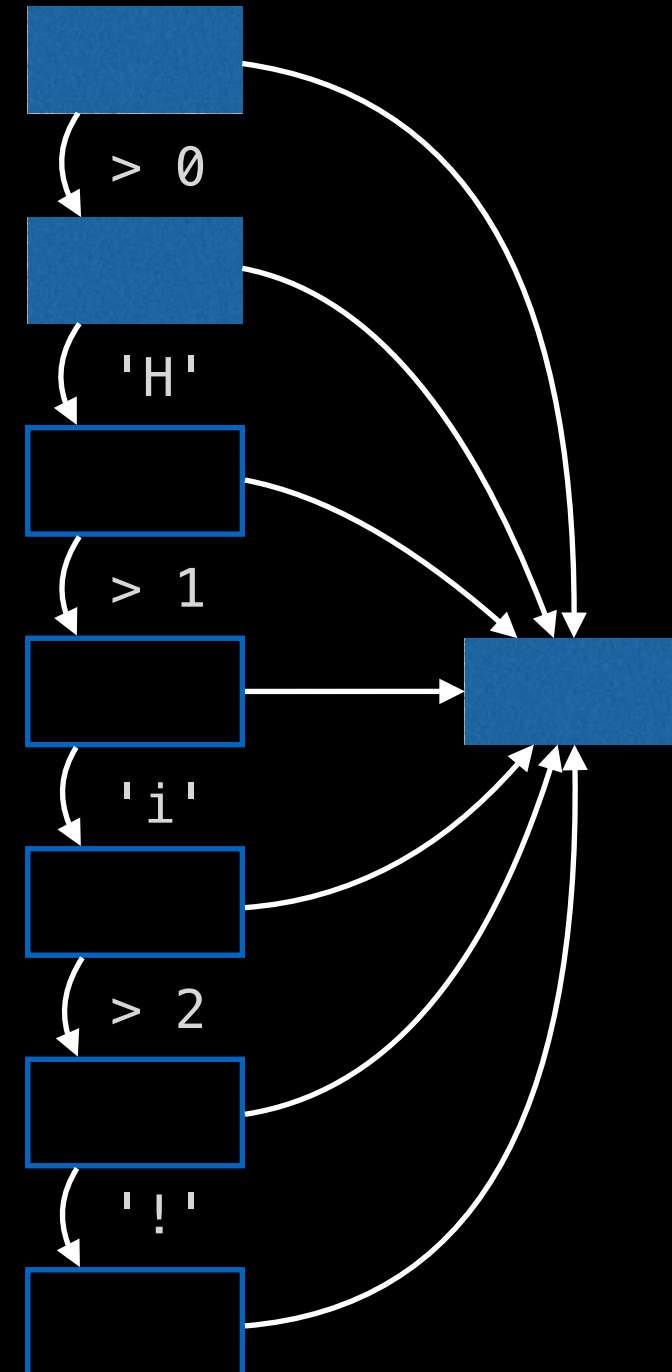


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: `<empty>`

Mutations: `q`

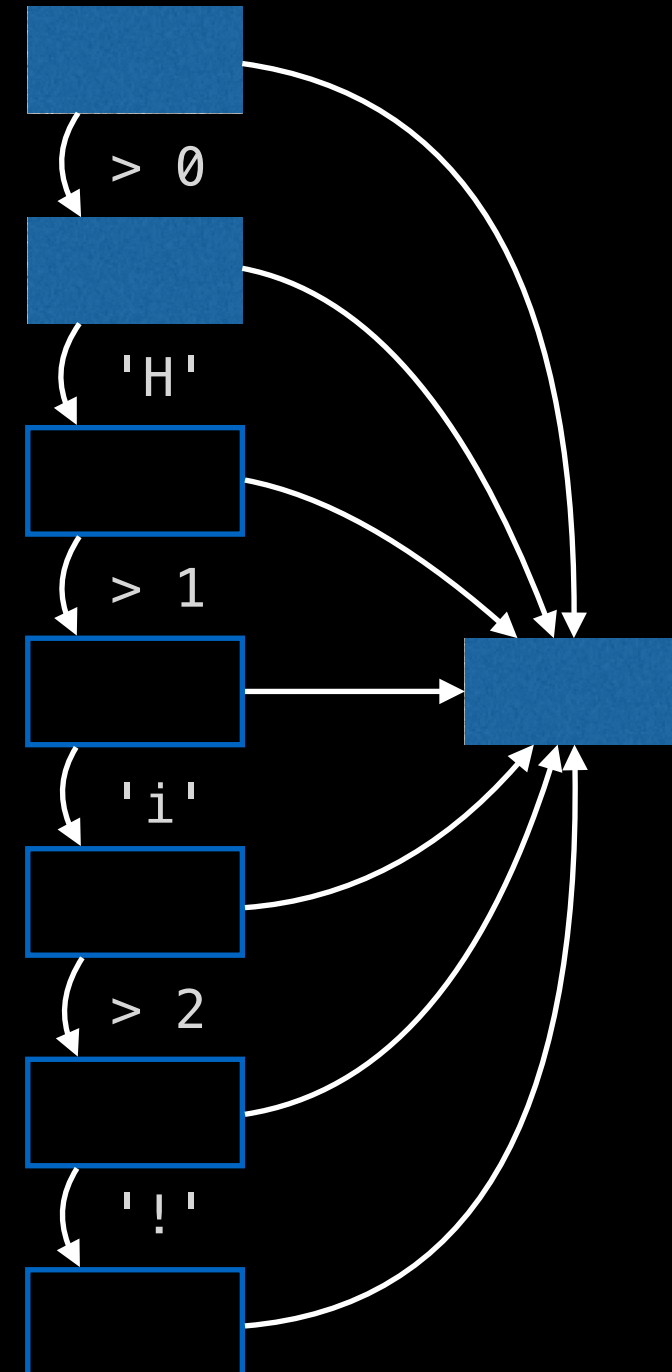
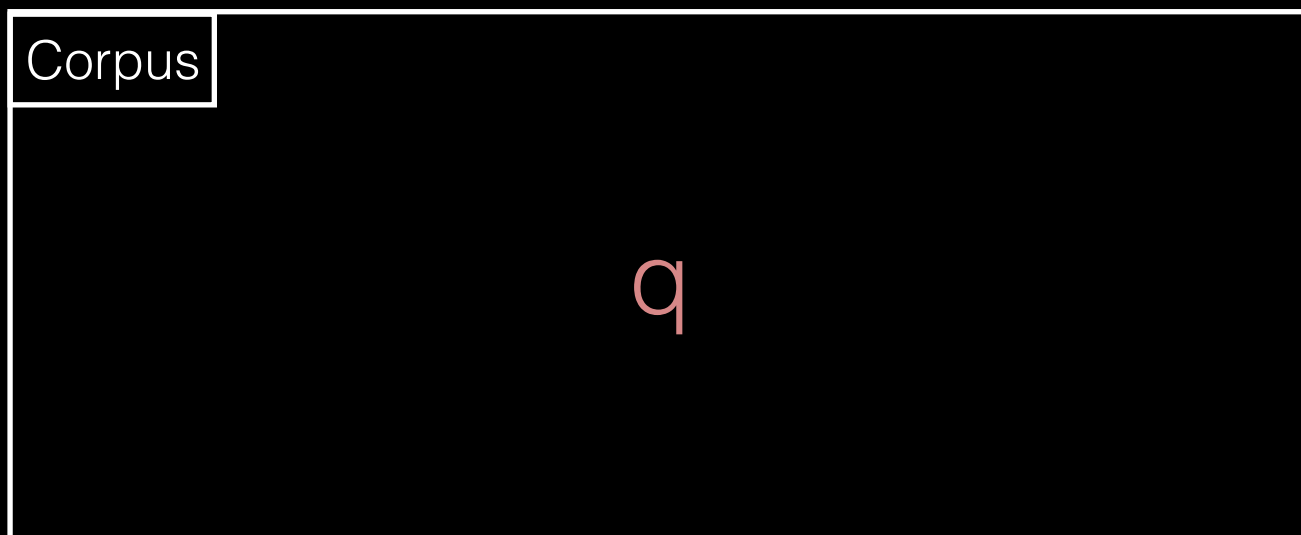


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: `<empty>`

Mutations: `q X 7 y`

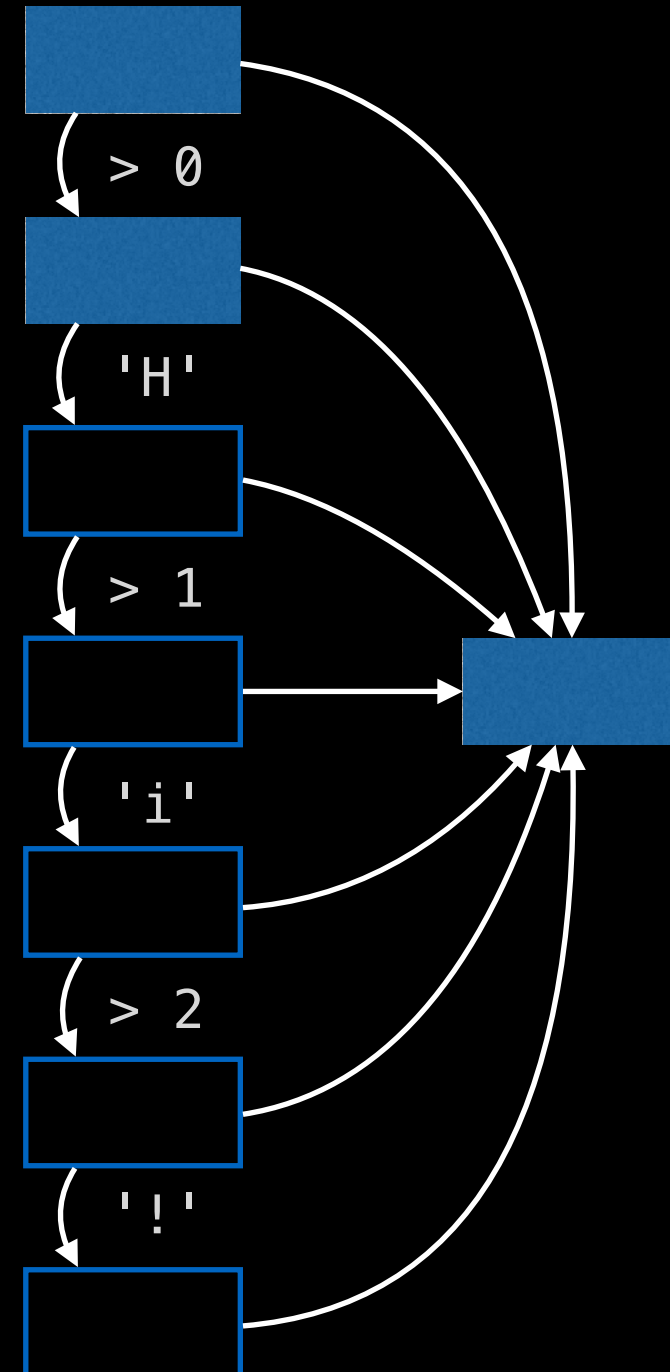
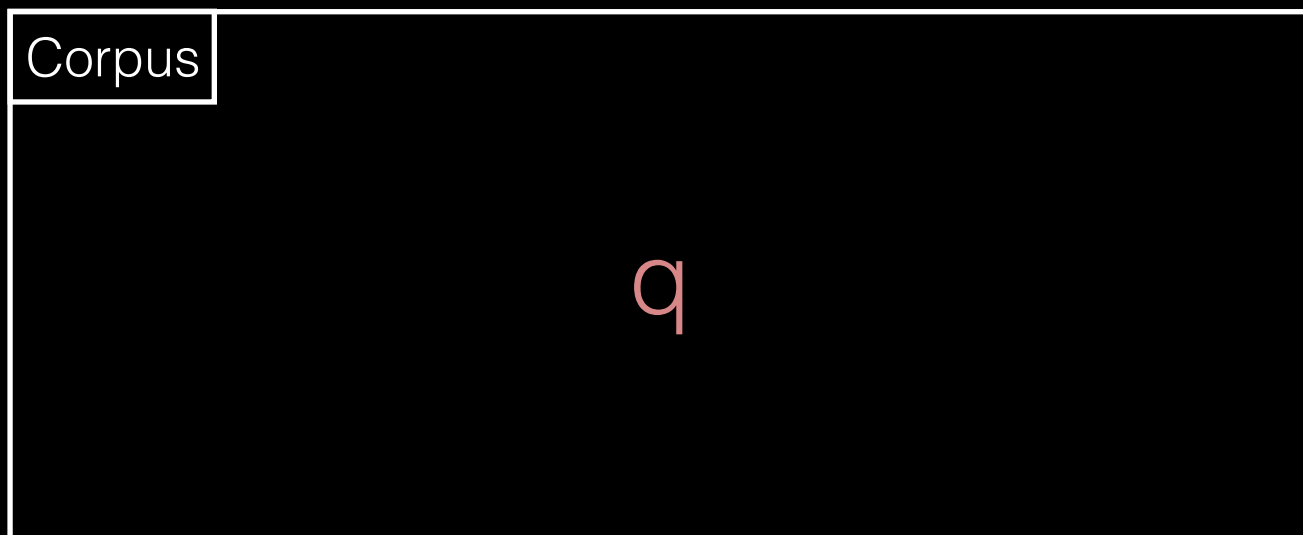


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: **q**

Mutations: **qZ y**

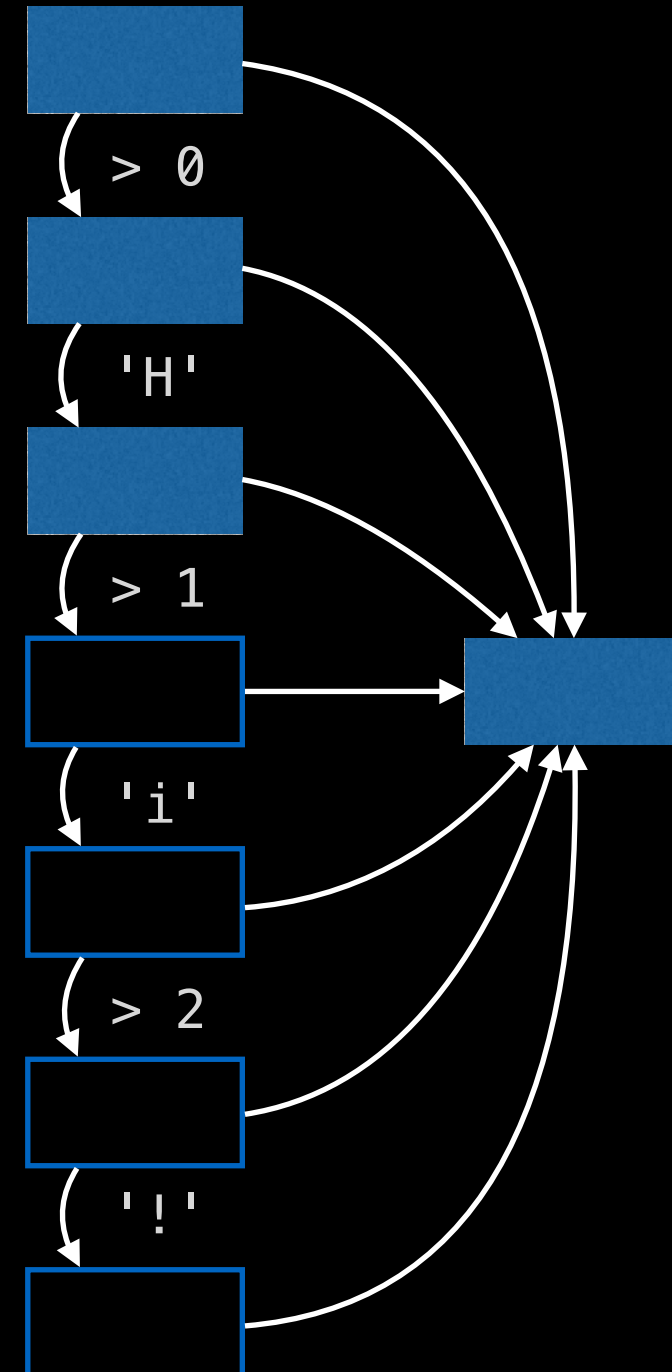
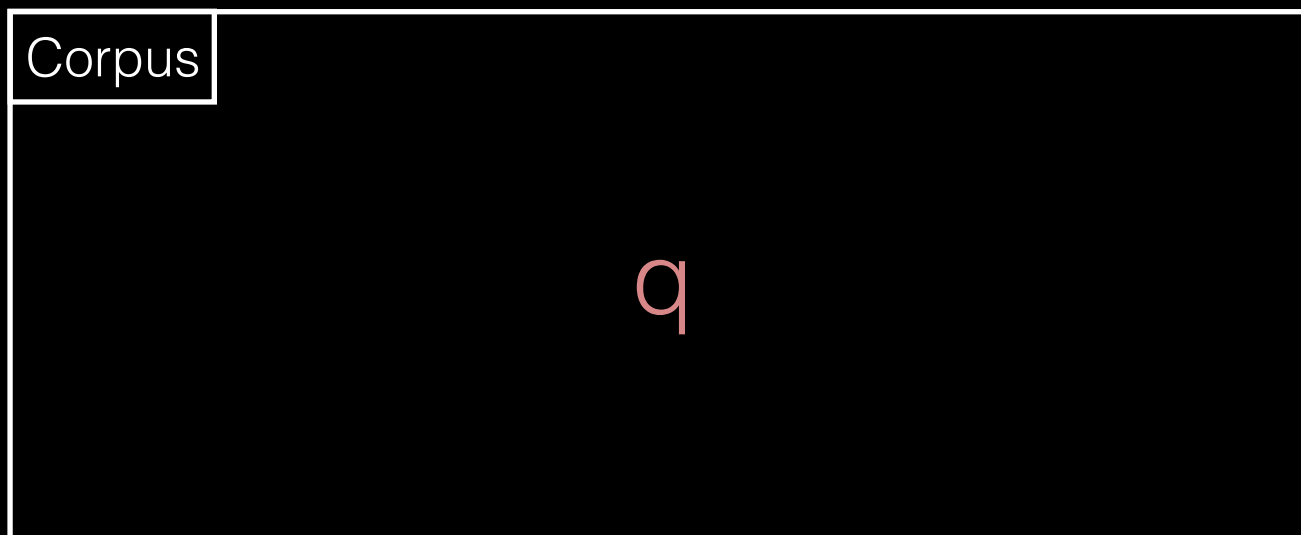


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: **q**

Mutations: **qZ y H**

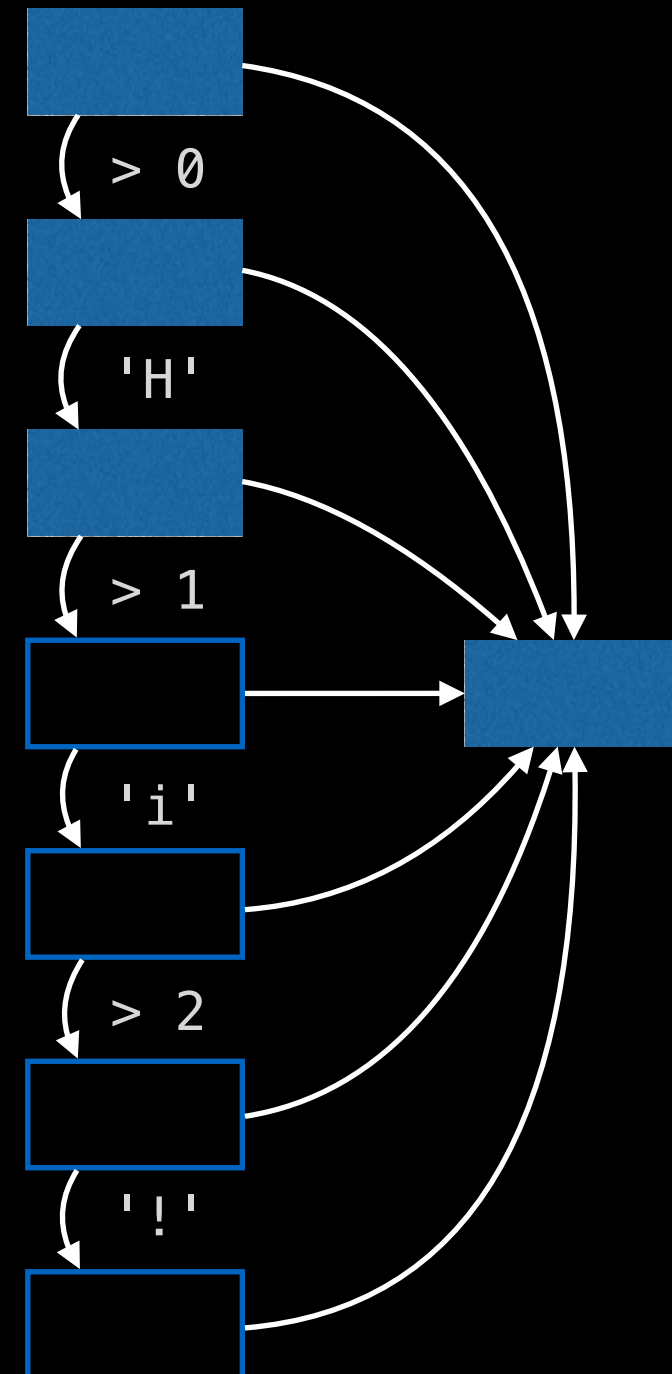
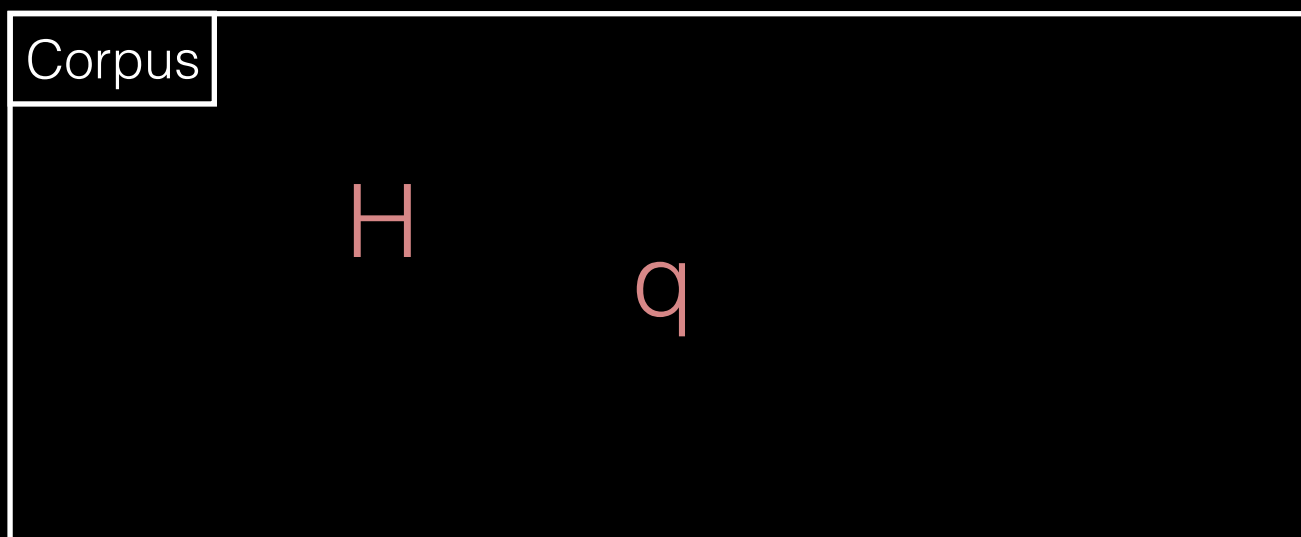


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: **q**

Mutations: **qZ y H qm**

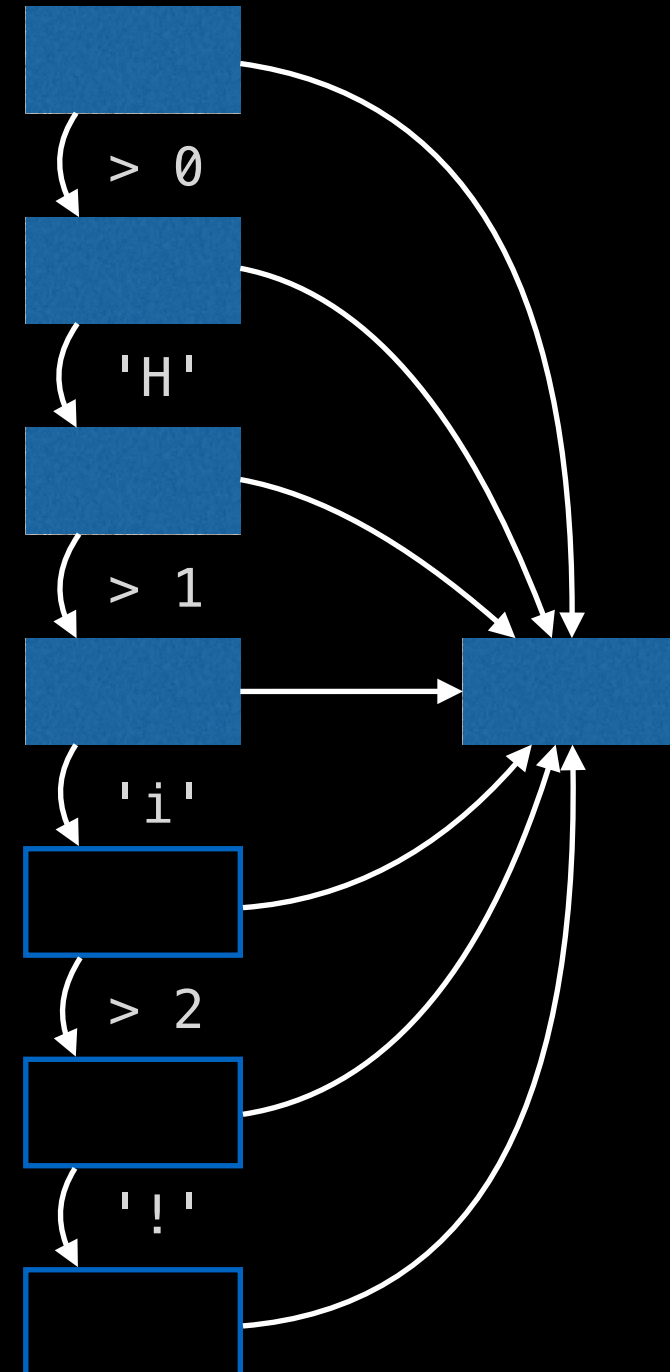
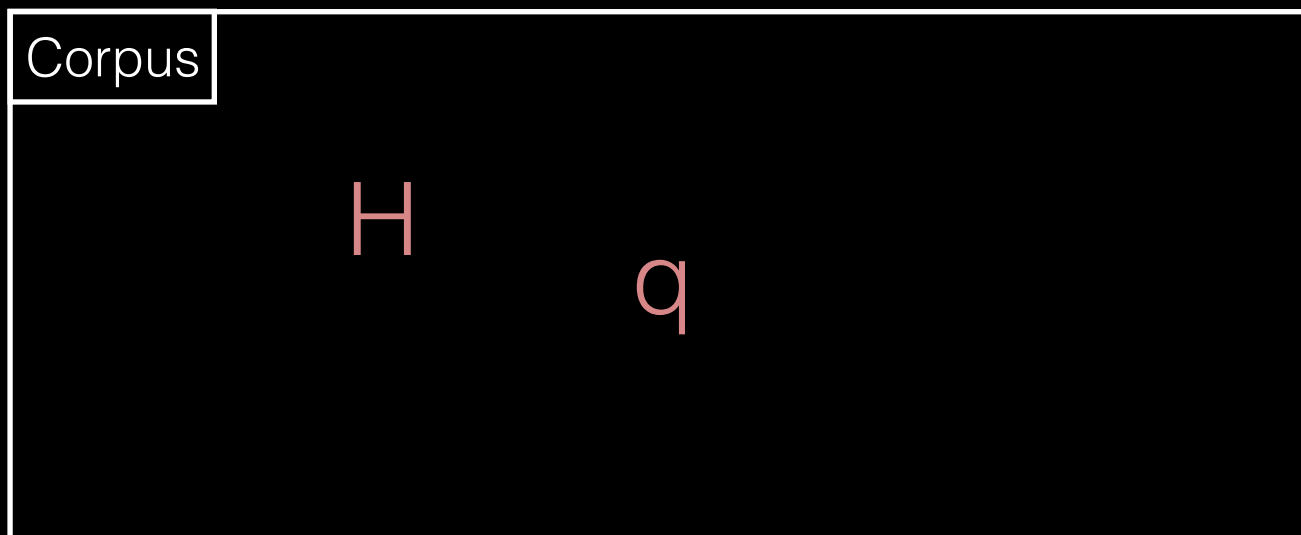


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: **H**

Mutations: **HF**

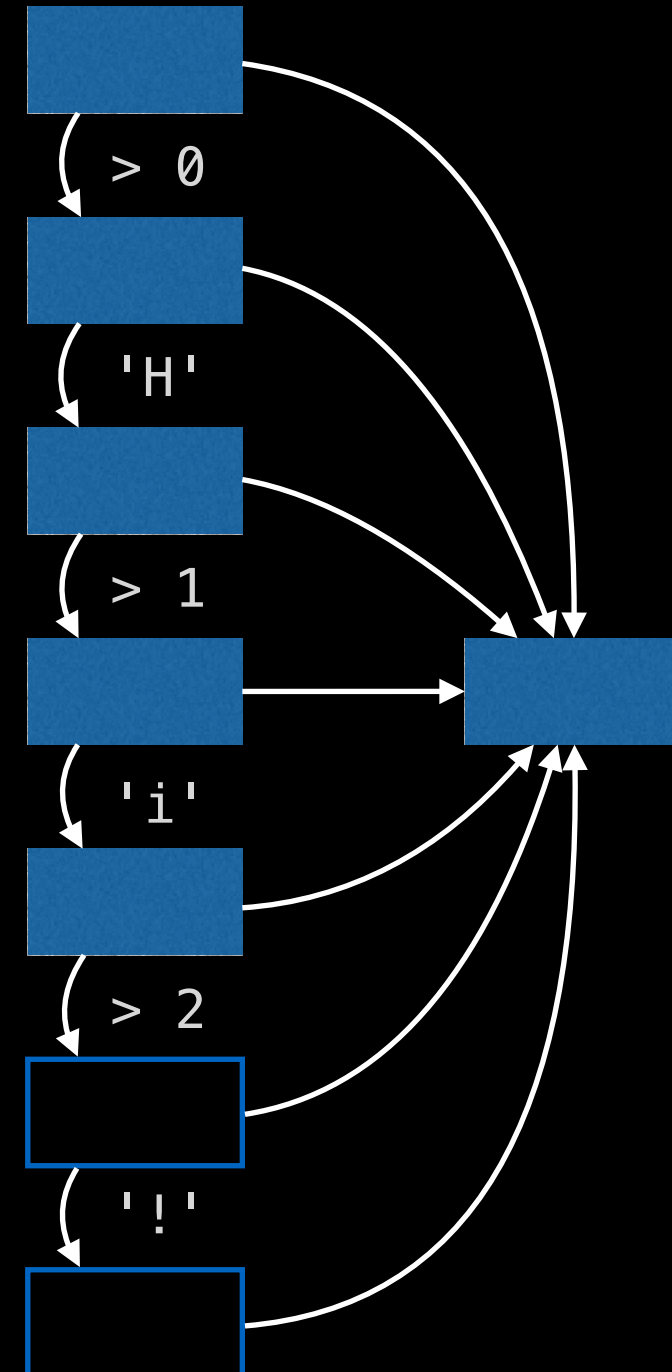
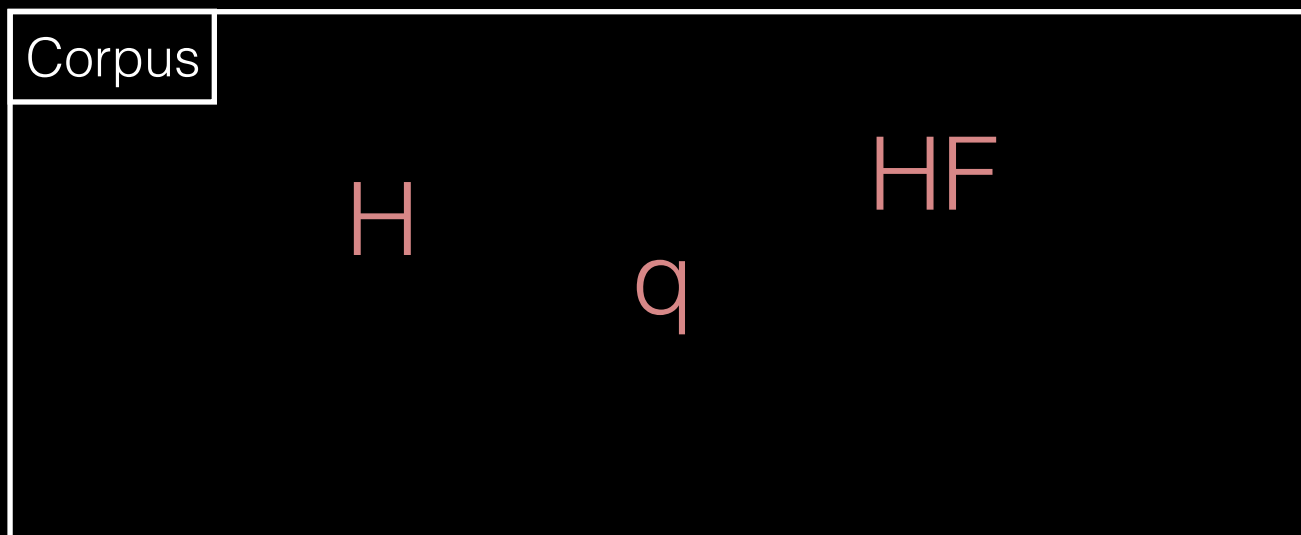


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: H

Mutations: HF Hi

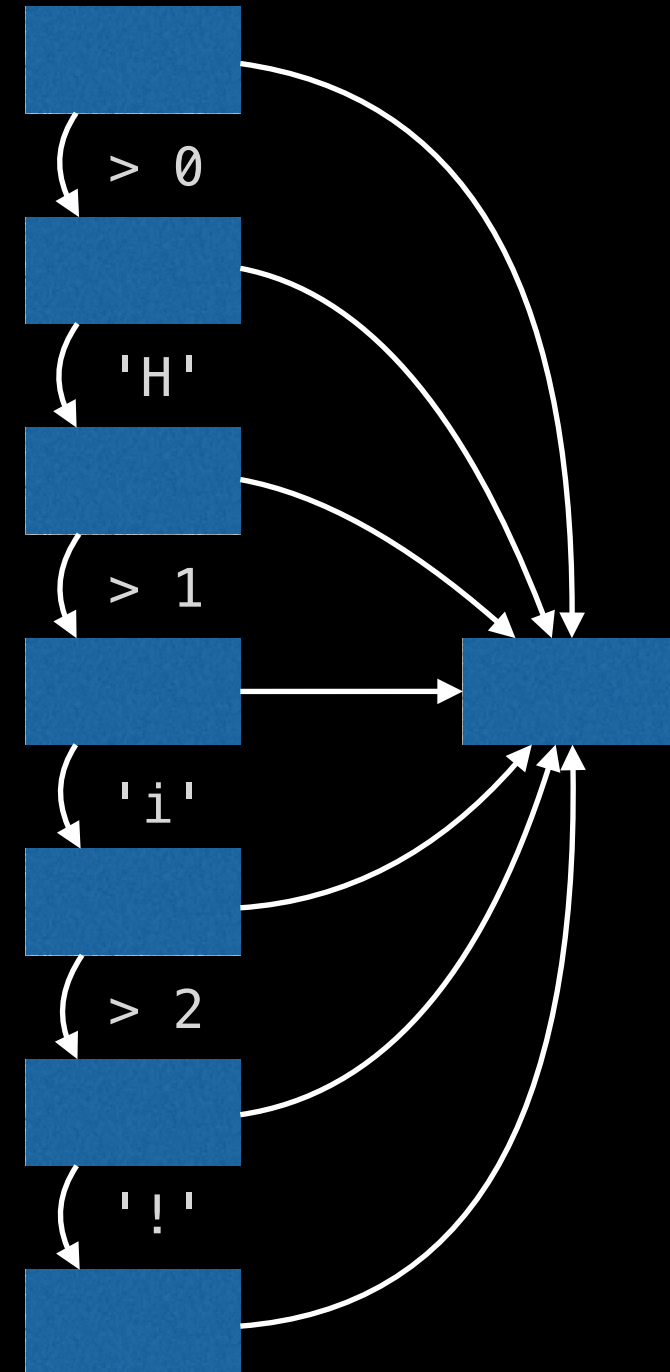
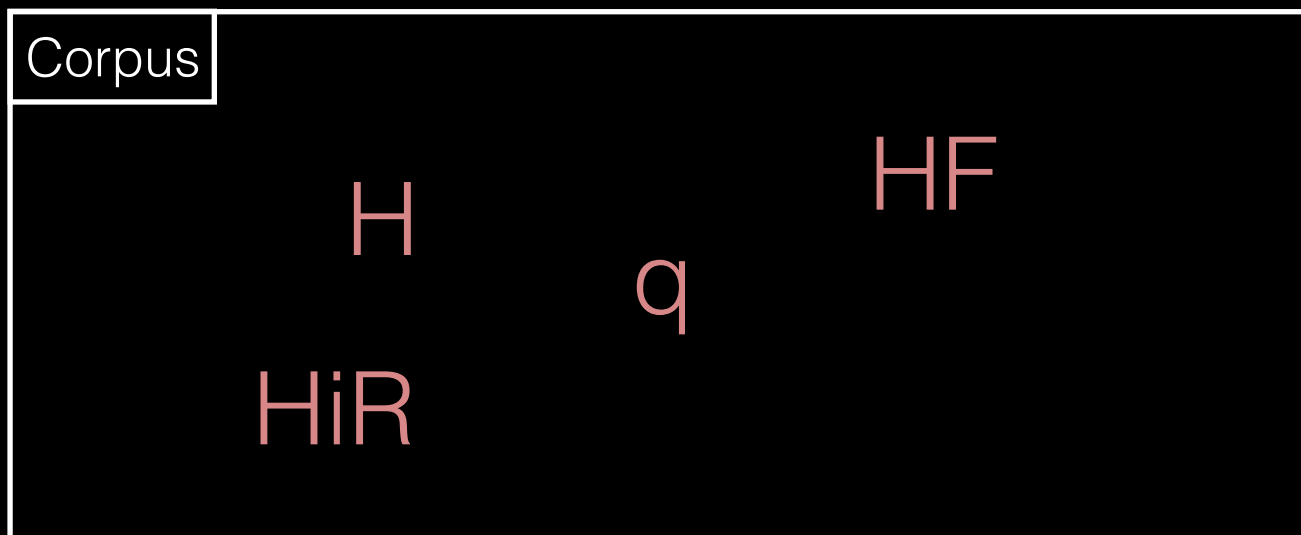


libFuzzer

```
extern "C" int LLVMFuzzerTestOneInput(  
  const uint8_t *Data, size_t Size) {  
  if (Size > 0 && Data[0] == 'H')  
    if (Size > 1 && Data[1] == 'i')  
      if (Size > 2 && Data[2] == '!')  
        exit(0);  
  return 0;  
}
```

Unit: **Hi**

Mutations: **Hh xi HiR Hi!**





Fuzzers in LLVM

- clang-fuzzer
- clang-format-fuzzer
- llvm-as-fuzzer
- llvm-mc-fuzzer
- ... but no llc-fuzzer


Beyond Parser Bugs

```
static void g(){}  
signed*Qwchar_t;  
overridedouble++!=~;inline-=}y=^bitand{;*or;goto*&&k}==n  
int XS/=~char16_t&s<=const_cast<Xchar*>(thread_local3+=char32_t
```


Beyond Parser Bugs


```
@a2 = global i8 addrspace(1)*@0 = private constant i32 0 @1 =  
private constant i32 1  
@2 = private alias i32* @d0  
@3 = @a  
@a = ad private aeflias i32* @1ine internal h2dden vodrsid @fun  
ction() {  
entry:  
ret void  
}pac  
e(1) global i8 0
```

Structured Fuzzing



```
00000000 64 65 66 69 6e 65 20 76 6f 69 64 20 40 66 28 29 |define void @f(|
00000010 20 7b 0a 42 42 3a 0a 20 20 25 4c 32 20 3d 20 6c | {.BB:. %L2 = l|
-00000020 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 65 |load i1, i1* unde|
+00000020 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 65 |load i8, i8* unde|
00000030 66 0a 20 20 62 72 20 6c 61 62 65 6c 20 25 42 42 |f. br label %BB|
00000040 35 0a 0a 42 42 39 3a 0a 20 20 25 4c 36 20 3d 20 |5..BB9:. %L6 = |
-00000050 6c 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 |load i1, i1* und|
+00000050 6c 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 |load i8, i8* und|
00000060 65 66 0a 20 20 25 42 38 20 3d 20 73 64 69 76 20 |ef. %B8 = sdiv |
-00000070 69 31 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i1 %L6, %L2. %A|
+00000070 69 38 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i8 %L6, %L2. %A|
00000080 37 20 3d 20 61 6c 6c 6f 63 61 20 66 6c 6f 61 74 |7 = alloca float|
00000090 0a 20 20 25 41 34 20 3d 20 61 6c 6c 6f 63 61 20 |. %A4 = alloca |
```

Structured Fuzzing



```
00000000 64 65 66 69 6e 65 20 76 6f 69 64 20 40 66 28 29 |define void @f(|
00000010 20 7b 0a 42 42 3a 0a 20 20 25 4c 32 20 3d 20 6c | {.BB:. %L2 = l|
-00000020 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 65 |load i1, i1* unde|
+00000020 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 65 |load i8, i8* unde|
00000030 66 0a 20 20 62 72 20 6c 61 62 65 6c 20 25 42 42 |f. br label %BB|
00000040 35 0a 0a 42 42 39 3a 0a 20 20 25 4c 36 20 3d 20 |5..BB9:. %L6 = |
-00000050 6c 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 |load i1, i1* und|
+00000050 6c 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 |load i8, i8* und|
00000060 65 66 0a 20 20 25 42 38 20 3d 20 73 64 69 76 20 |ef. %B8 = sdiv |
-00000070 69 31 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i1 %L6, %L2. %A|
+00000070 69 38 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i8 %L6, %L2. %A|
00000080 37 20 3d 20 61 6c 6c 6f 63 61 20 66 6c 6f 61 74 |7 = alloca float|
00000090 0a 20 20 25 41 34 20 3d 20 61 6c 6c 6f 63 61 20 |. %A4 = alloca |
```

Structured Fuzzing

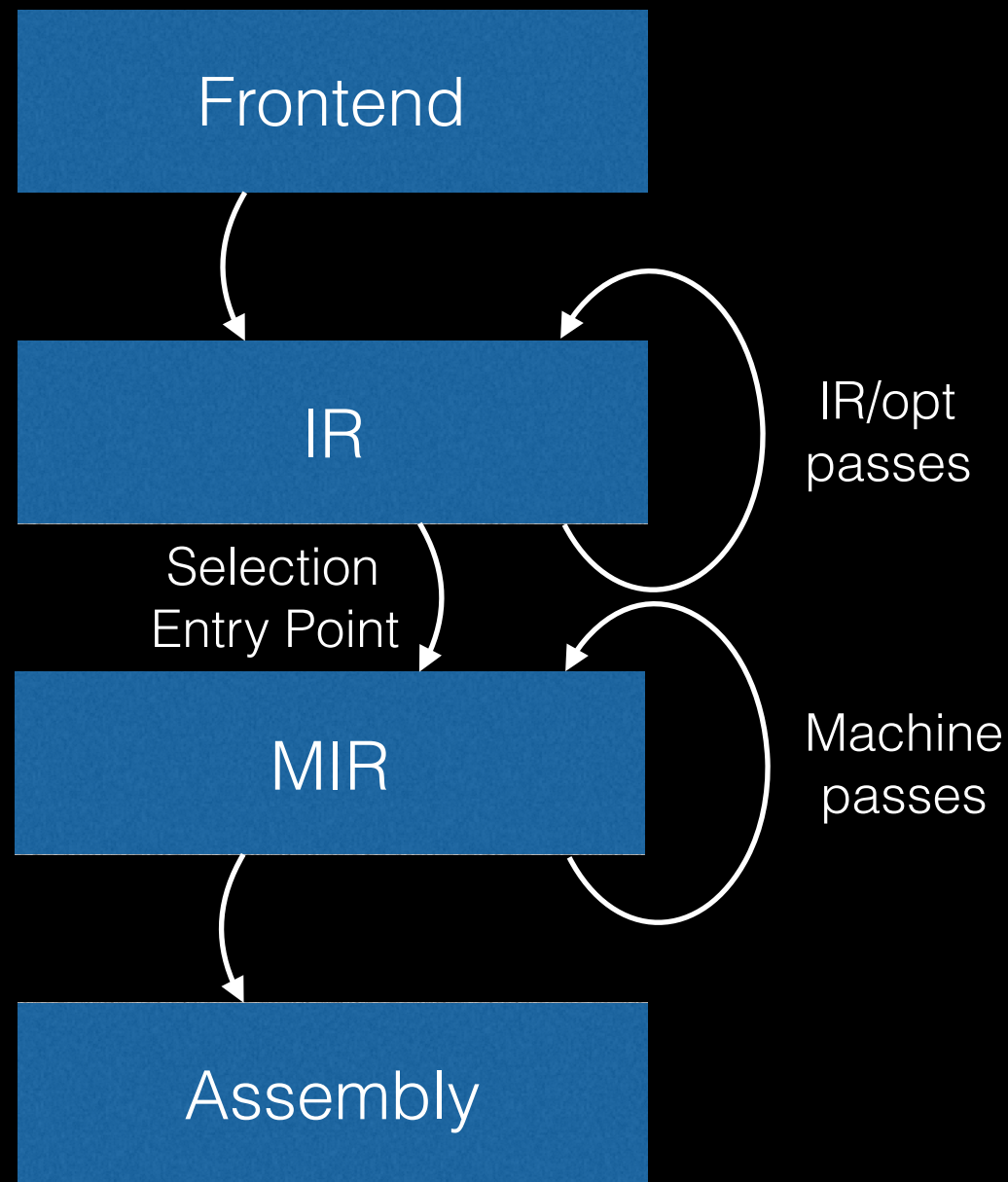
```
00000000 64 65 66 69 6e 65 20 76 6f 69 64 20 40 66 28 29 |define void @f(|
00000010 20 7b 0a 42 42 3a 0a 20 20 25 4c 32 20 3d 20 6c | {.BB:. %L2 = l|
-00000020 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 65 |load i1, i1* unde|
+00000020 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 65 |load i8, i8* unde|
00000030 66 0a 20 20 62 72 20 6c 61 62 65 6c 20 25 42 42 |f. br label %BB|
00000040 35 0a 0a 42 42 39 3a 0a 20 20 25 4c 36 20 3d 20 |5..BB9:. %L6 = |
-00000050 6c 6f 61 64 20 69 31 2c 20 69 31 2a 20 75 6e 64 |load i1, i1* und|
+00000050 6c 6f 61 64 20 69 38 2c 20 69 38 2a 20 75 6e 64 |load i8, i8* und|
00000060 65 66 0a 20 20 25 42 38 20 3d 20 73 64 69 76 20 |ef. %B8 = sdiv |
-00000070 69 31 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i1 %L6, %L2. %A|
+00000070 69 38 20 25 4c 36 2c 20 25 4c 32 0a 20 20 25 41 |i8 %L6, %L2. %A|
00000080 37 20 3d 20 61 6c 6c 6f 63 61 20 66 6c 6f 61 74 |7 = alloca float|
00000090 0a 20 20 25 41 34 20 3d 20 61 6c 6c 6f 63 61 20 |. %A4 = alloca |
```

Custom Mutator API



```
// Optional user-provided custom mutator.  
// Mutates raw data in [Data, Data+Size) inplace.  
// Returns the new size, which is not greater than MaxSize.  
// Given the same Seed produces the same mutation.  
size_t LLVMFuzzerCustomMutator(uint8_t *Data,  
                                size_t Size,  
                                size_t MaxSize,  
                                unsigned int Seed);
```

Where to Mutate?





llvm-stress

- Random IR generator
- Used for new backends and FastISel
- Excellent for bringup, forgotten later



LLVM IR Mutator

- Work in terms of operations on SSA values
- Each operation has sources and a sink
- Sinks should be side effects to avoid dead code
- It's safe to disconnect sinks and DCE at will

Adding one operation

- Divide a block into potential sources and sinks

```
define void @f() {  
BB:  
  %L1 = load i16, i16* undef  
  %L2 = load i16, i16* undef  
  %B1 = lshr i16 %L1, %L2  
  store i16 %B1, i16* undef  
  ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks

```
define void @f() {  
BB:  
    %L1 = load i16, i16* undef  
    %L2 = load i16, i16* undef  


---

  
    %B1 = lshr i16 %L1, %L2  
    store i16 %B1, i16* undef  
    ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source

```
define void @f() {  
  BB:  
  → %L1 = load i16, i16* undef  
    %L2 = load i16, i16* undef  
  

---

  
    %B1 = lshr i16 %L1, %L2  
    store i16 %B1, i16* undef  
    ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source
- Choose an operation

```
define void @f() {  
BB:  
  %L1 = load i16, i16* undef  
  %L2 = load i16, i16* undef  
  %B2 = add i16 %L1, ?  
  %B1 = lshr i16 %L1, %L2  
  store i16 %B1, i16* undef  
  ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source
- Choose an operation
- Fill in the other sources

```
define void @f() {  
  BB:  
  → %L1 = load i16, i16* undef  
    %L2 = load i16, i16* undef  
    %B2 = add i16 %L1, %L1  
    %B1 = lshr i16 %L1, %L2  
    store i16 %B1, i16* undef  
    ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source
- Choose an operation
- Fill in the other sources
- Steal an argument of a later operation as a sink

```
define void @f() {  
BB:  
  %L1 = load i16, i16* undef  
  %L2 = load i16, i16* undef  
  %B2 = add i16 %L1, %L1  
  %B1 = lshr i16 %L1, %L2 ←  
  store i16 %B1, i16* undef  
  ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source
- Choose an operation
- Fill in the other sources
- Steal an argument of a later operation as a sink

```
define void @f() {  
BB:  
  %L1 = load i16, i16* undef  
  %L2 = load i16, i16* undef  
  %B2 = add i16 %L1, %L1  
  %B1 = lshr i16 %L1, %B2  
  store i16 %B1, i16* undef  
  ret void  
}
```

Adding one operation

- Divide a block into potential sources and sinks
- Choose or create one source
- Choose an operation
- Fill in the other sources
- Steal an argument of a later operation as a sink
- Clean up dead code

```
define void @f() {  
BB:  
  %L1 = load i16, i16* undef  
  %B2 = add i16 %L1, %L1  
  %B1 = lshr i16 %L1, %B2  
  store i16 %B1, i16* undef  
  ret void  
}
```


Splitting Blocks

```
BB:  
%L7 = load i1, i1* undef  
br label %BB5
```

```
BB5:  
%L6 = load i1, i1* undef  
%B8 = sdiv i1 %L6, %L7  
%A7 = alloca float  
%A4 = alloca float  
%L5 = load float, float* %A4  
%A1 = alloca float  
%L3 = load float, float* %A1  
%L1 = load i32, i32* undef  
%B6 = frem float %L3, %L5  
%A = alloca i32  
%L = load i32, i32* %A  
%B = xor i32 %L, %L1  
%B1 = xor i32 %B, %B  
store i32 %B1, i32* %A  
store float %B6, float* %A7  
store i1 %B8, i1* undef  
ret void
```

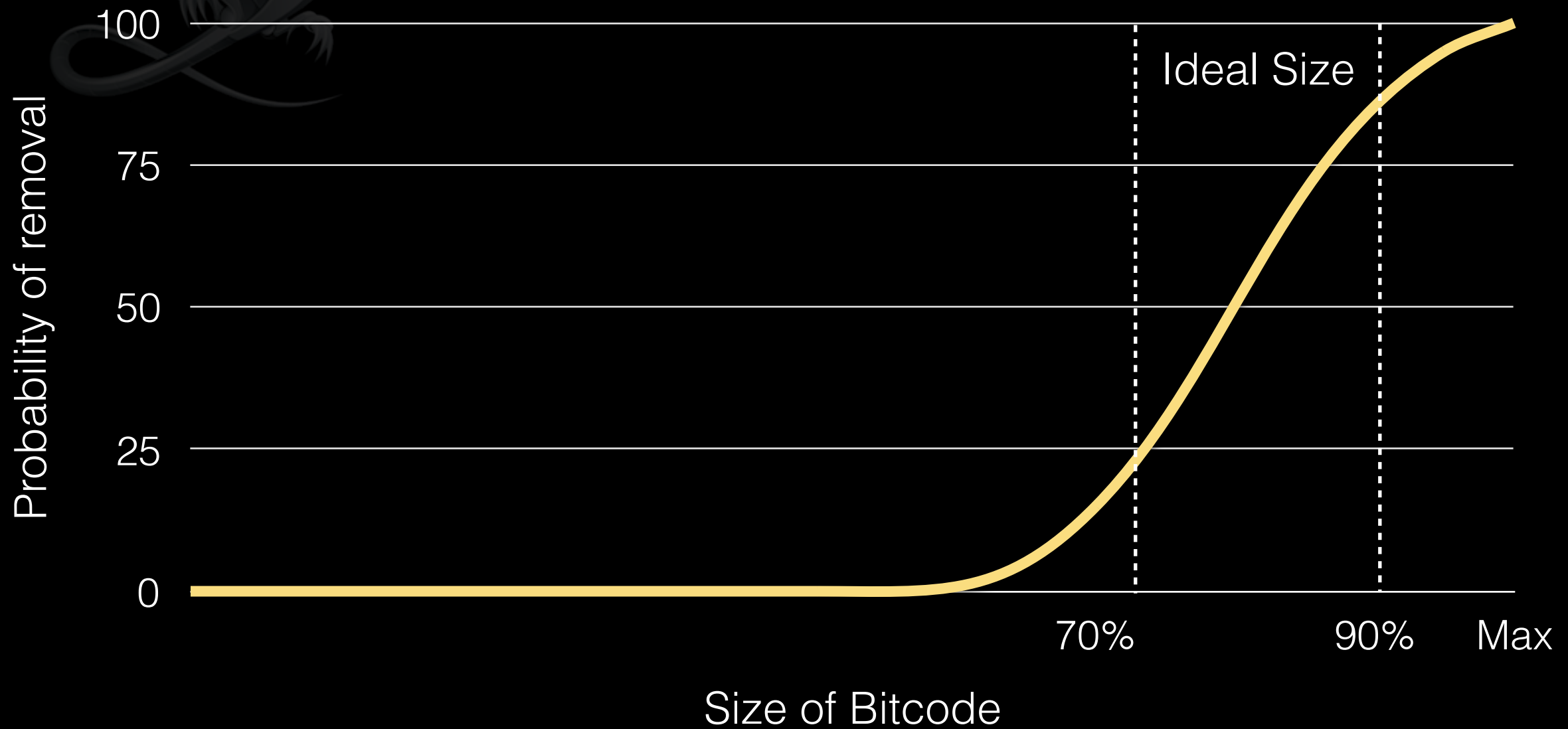
```
BB:  
%L7 = load i1, i1* undef  
br label %BB5
```

```
BB5:  
%L6 = load i1, i1* undef  
%B8 = sdiv i1 %L6, %L7  
%A7 = alloca float  
%A4 = alloca float  
br i1 %B8, label %BB5, label %BB1
```

```
BB1:  
%L5 = load float, float* %A4  
%A1 = alloca float  
%L3 = load float, float* %A1  
%L1 = load i32, i32* undef  
%B6 = frem float %L3, %L5  
%A = alloca i32  
%L = load i32, i32* %A  
%B = xor i32 %L, %L1  
%B1 = xor i32 %B, %B  
store i32 %B1, i32* %A  
store float %B6, float* %A7  
store i1 %B8, i1* undef  
ret void
```



Removing Code



Finding Bugs



By Sffubs (Own work) [[CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/)] via Wikimedia Commons



Correctness Checks

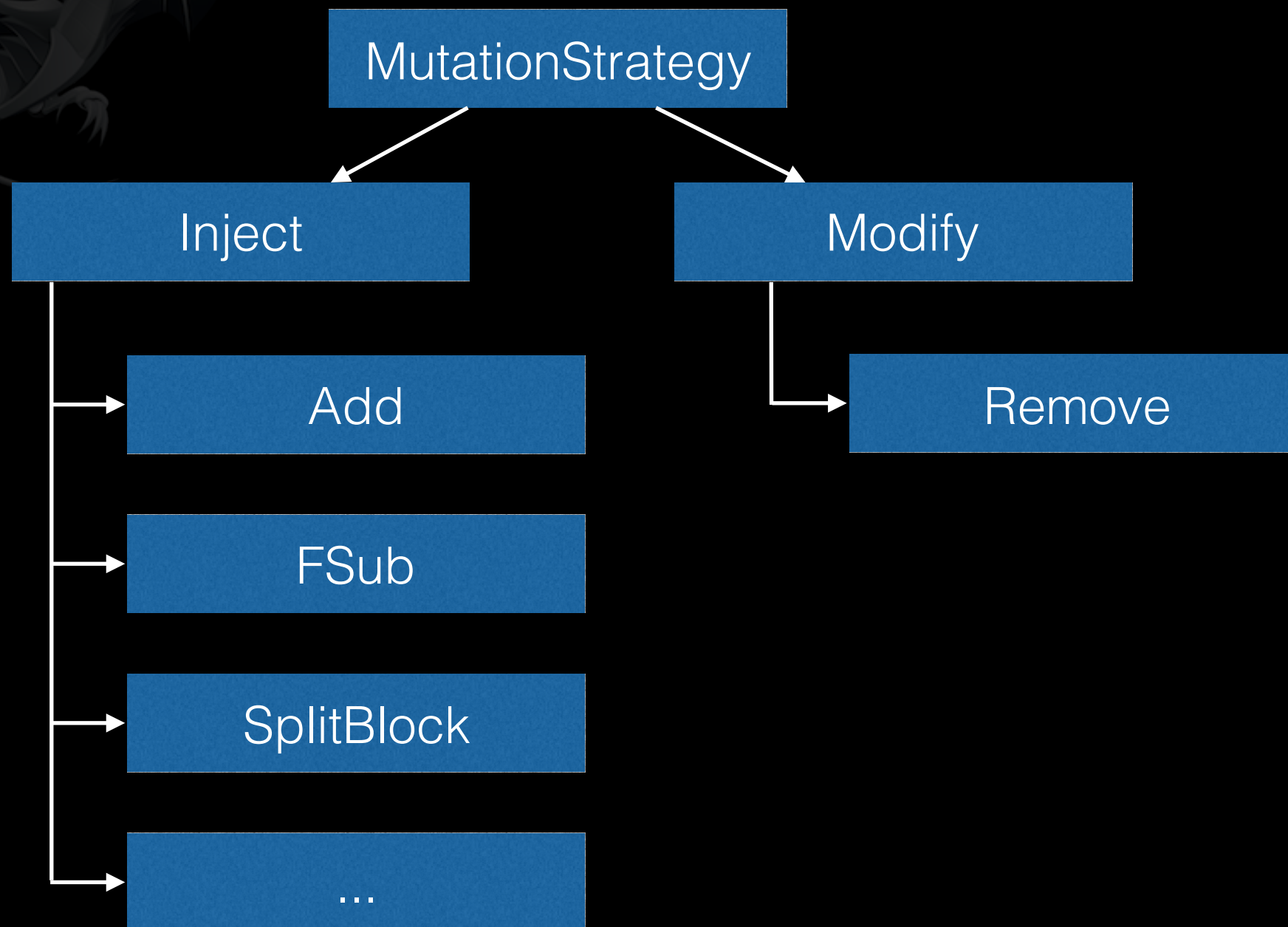
- Garbage in, garbage out
- Comparing instruction selectors
- Semantics-preserving mutations



Mutation Library

- Legal types are configurable
- New operations follow a simple interface
- Mix and match the operations you want

Mutation Library





Results

- Found various AArch64 GlobalSel bugs
- SelectionDAG is surprisingly hard to crash
- Test your own backends today
- IR mutator library available for other fuzzers



Questions?

