BinRec: Attack Surface Reduction Through Dynamic Binary Recovery

Taddeus Kroes, Anil Altinay, Joseph Nash, Yeoul Na, Stijn Volckaert, Herbert Bos, Michael Franz, Cristiano Giuffrida

October 19, 2018

VU VRIJE UNIVERSITEIT AMSTERDAM

UCIRVINE

Attack Surface Reduction

```
x = getenv("SET_ME");
if (x)
    cold_code();
hot_code();
```

Attack Surface Reduction

```
x = getenv("SET_ME");
if (x)
    cold_code();
hot_code();
```

Buggy features

ROP gadgets

Attack Surface Reduction

```
x = getenv("SET_ME");
```

~~`if (x)`~~

~~`    cold_code();`~~

Remove unwanted features

```
hot_code();
```

Attack surface reduced
to well-tested code

Attack Surface Reduction

```
setme_str: .asciz: “SET_ME”
push setme_str
call getenv
```

~~`cmp eax, 0`~~

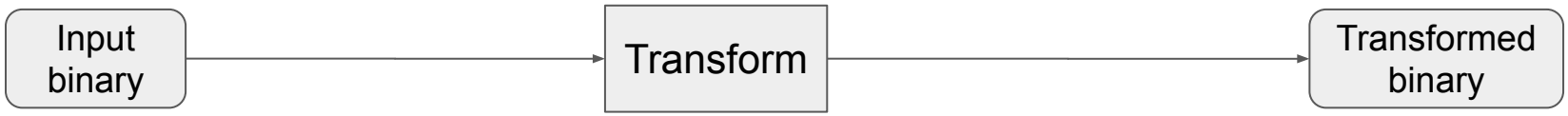
~~`je main`~~

~~`call cold_code`~~

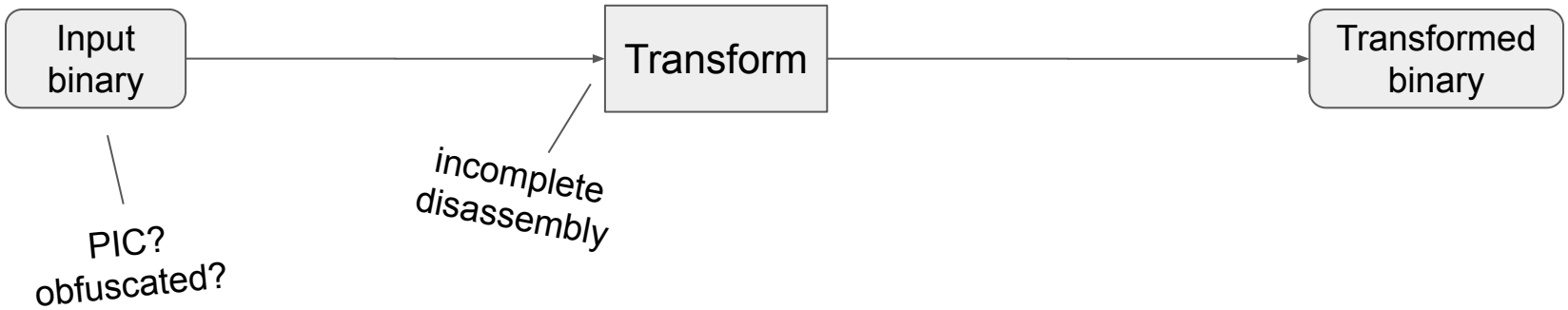
```
main:
call hot_code
```

Want to work on COTS binaries

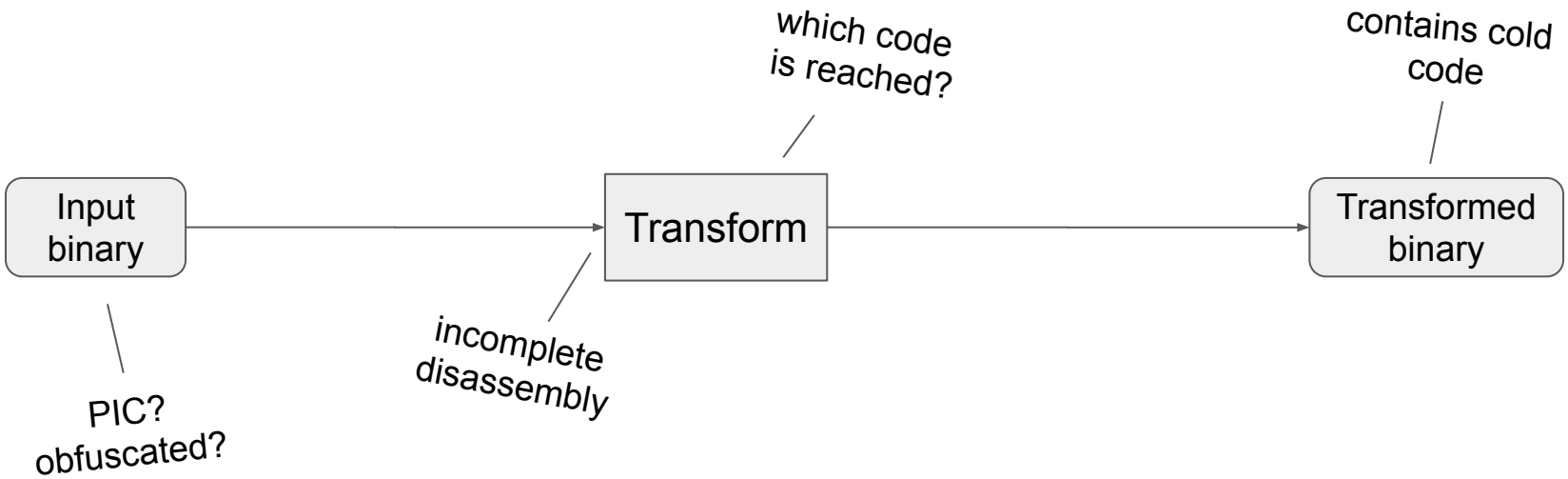
Static approach



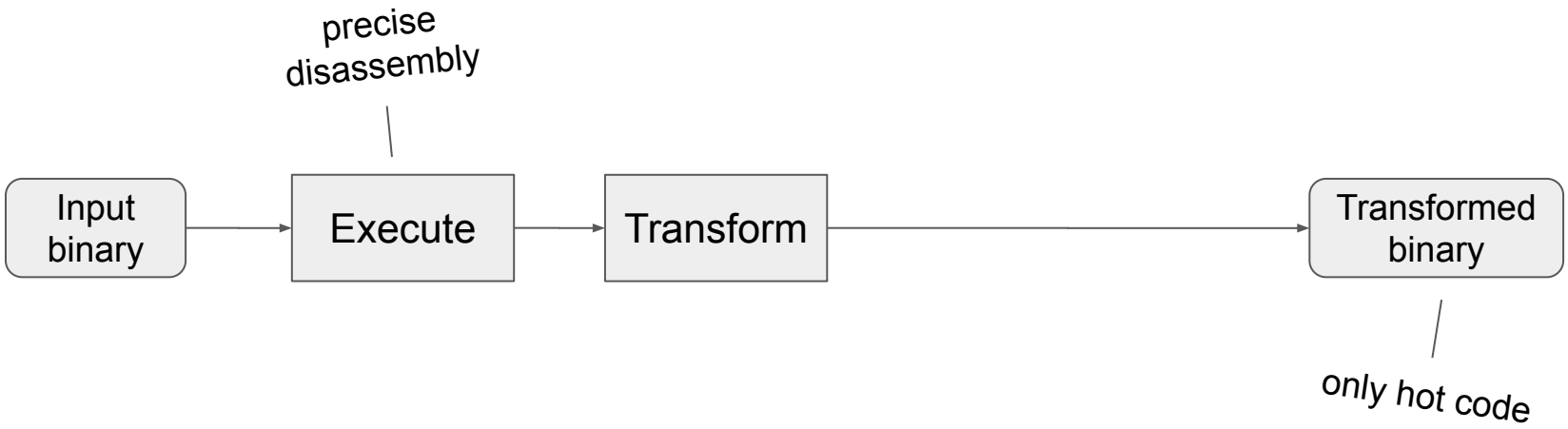
Static approach



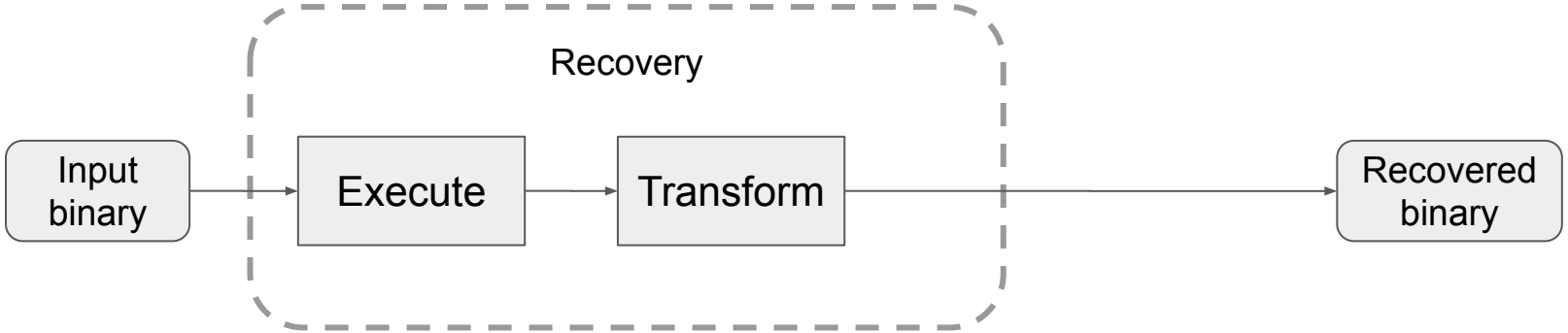
Static approach



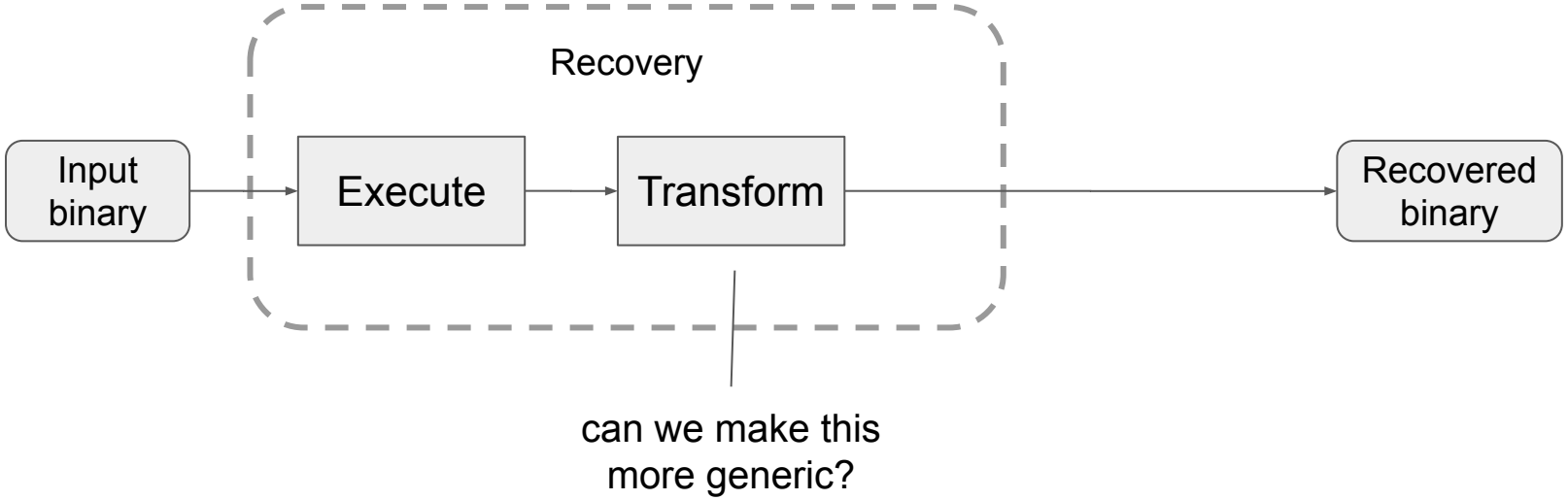
Dynamic approach by BinRec



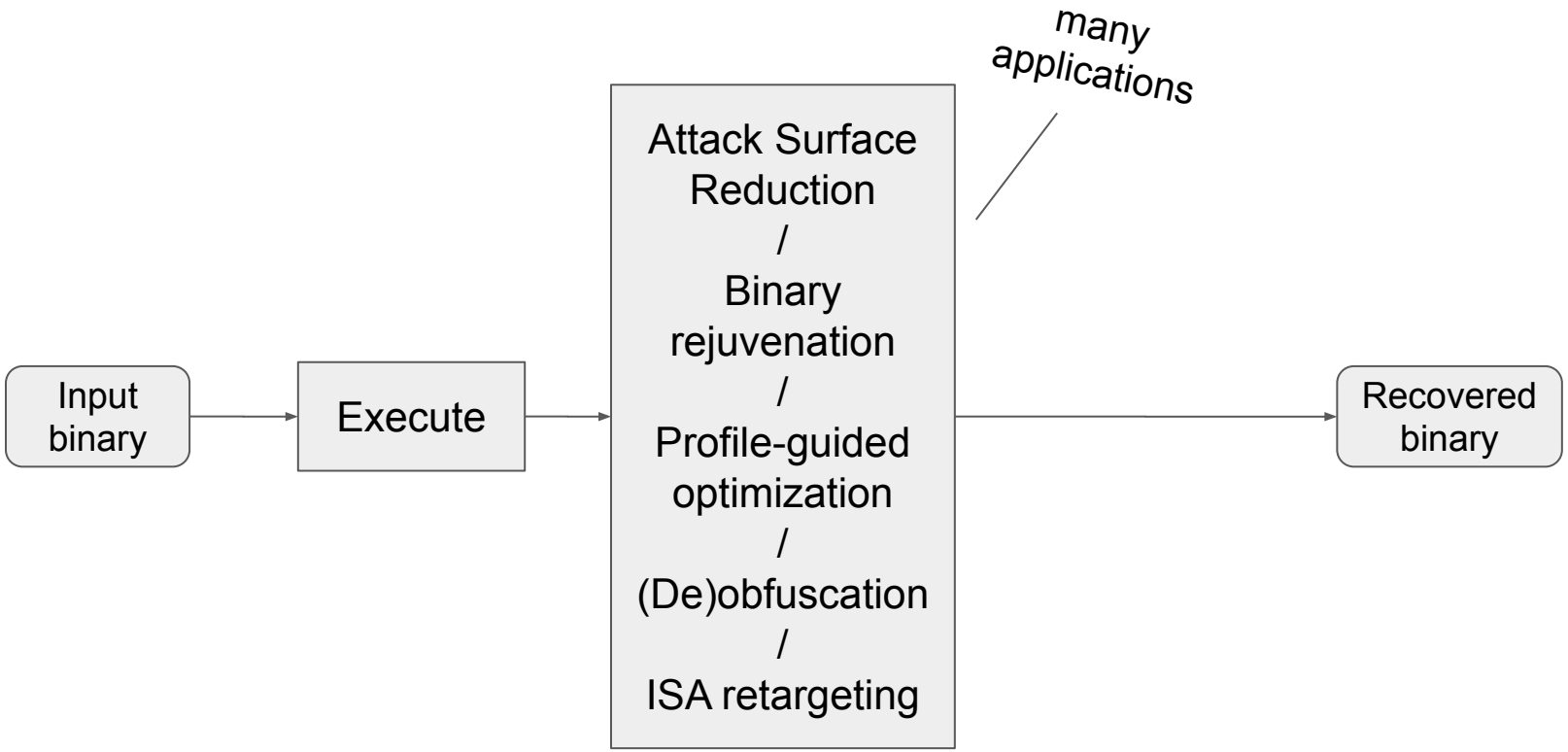
Dynamic approach by BinRec



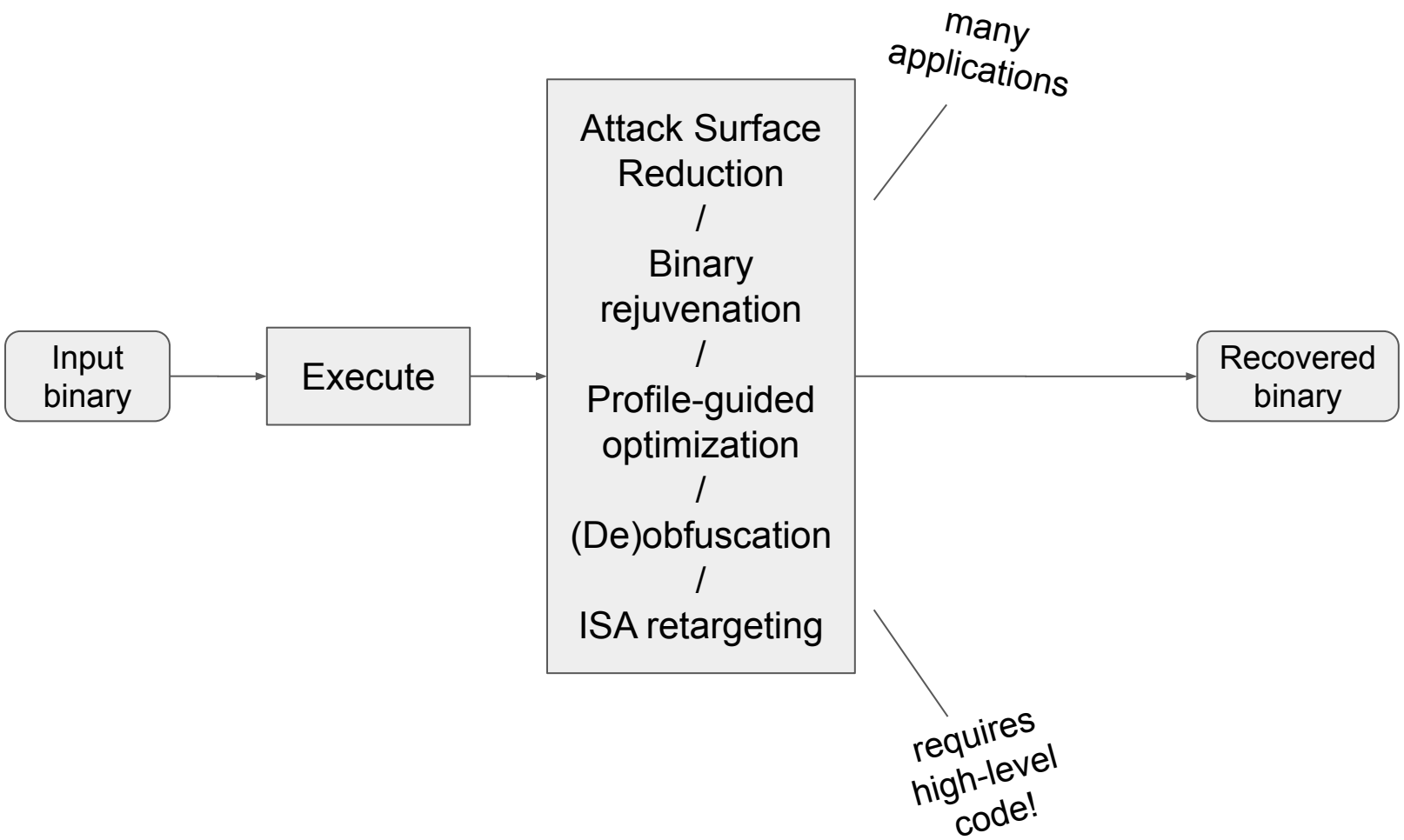
Dynamic approach by BinRec



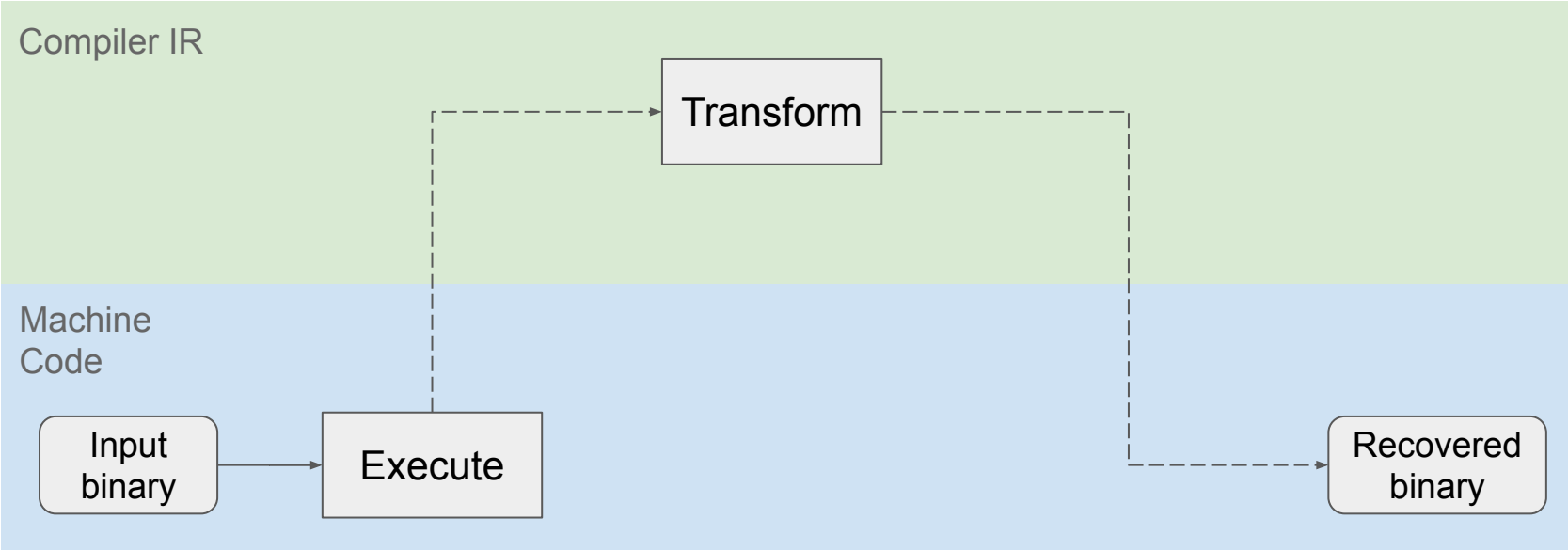
BinRec goal: complex binary transformation



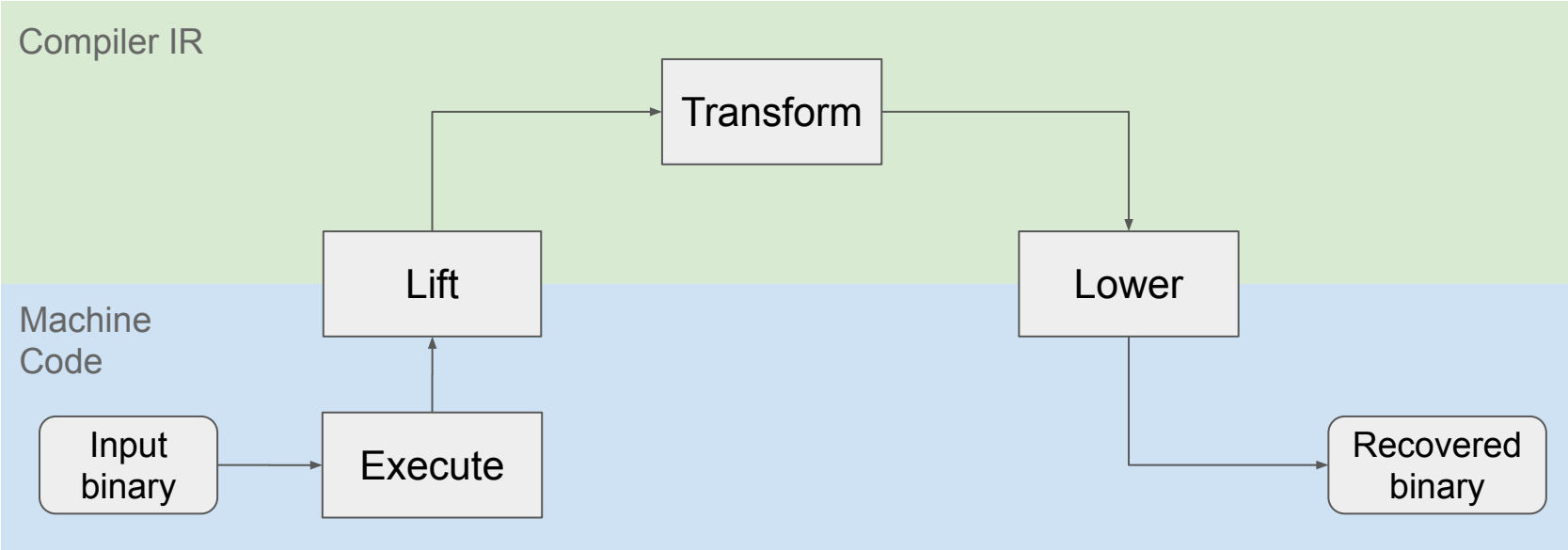
BinRec goal: complex binary transformation



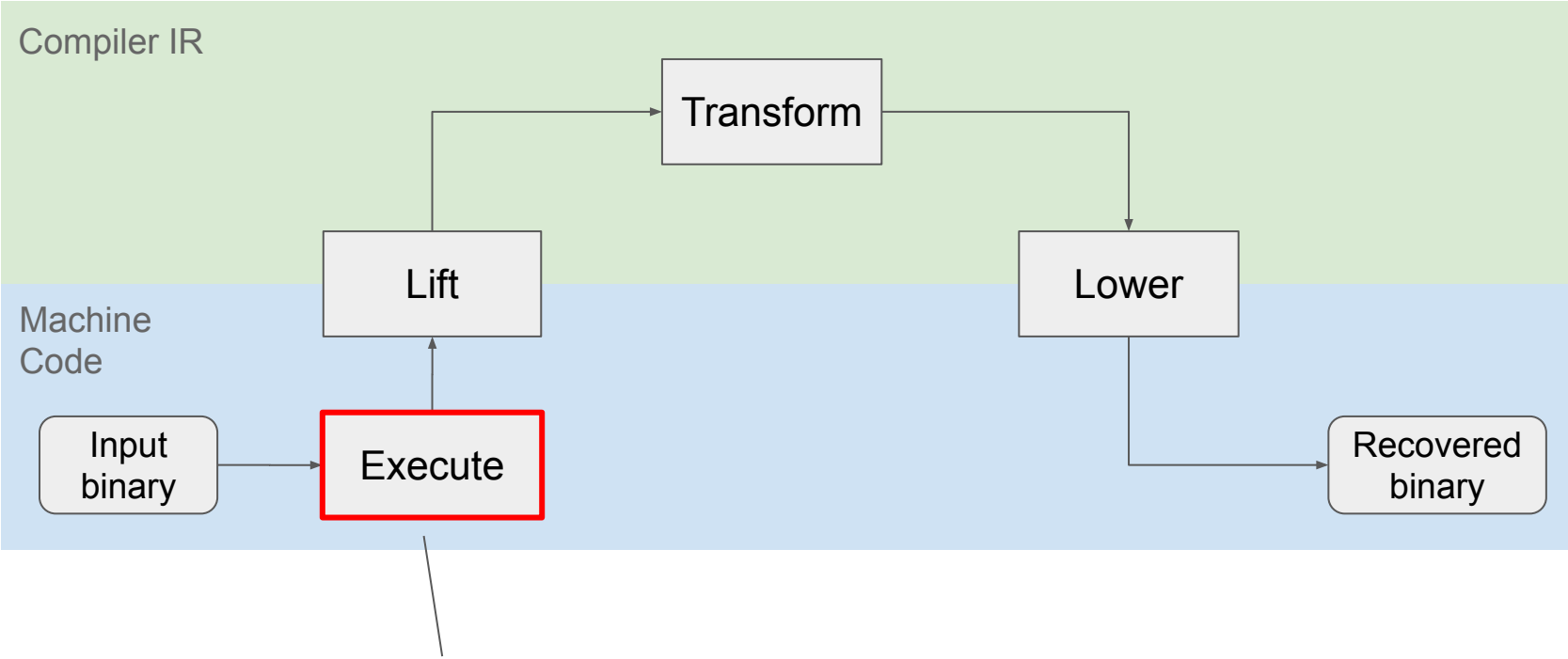
BinRec design



BinRec design

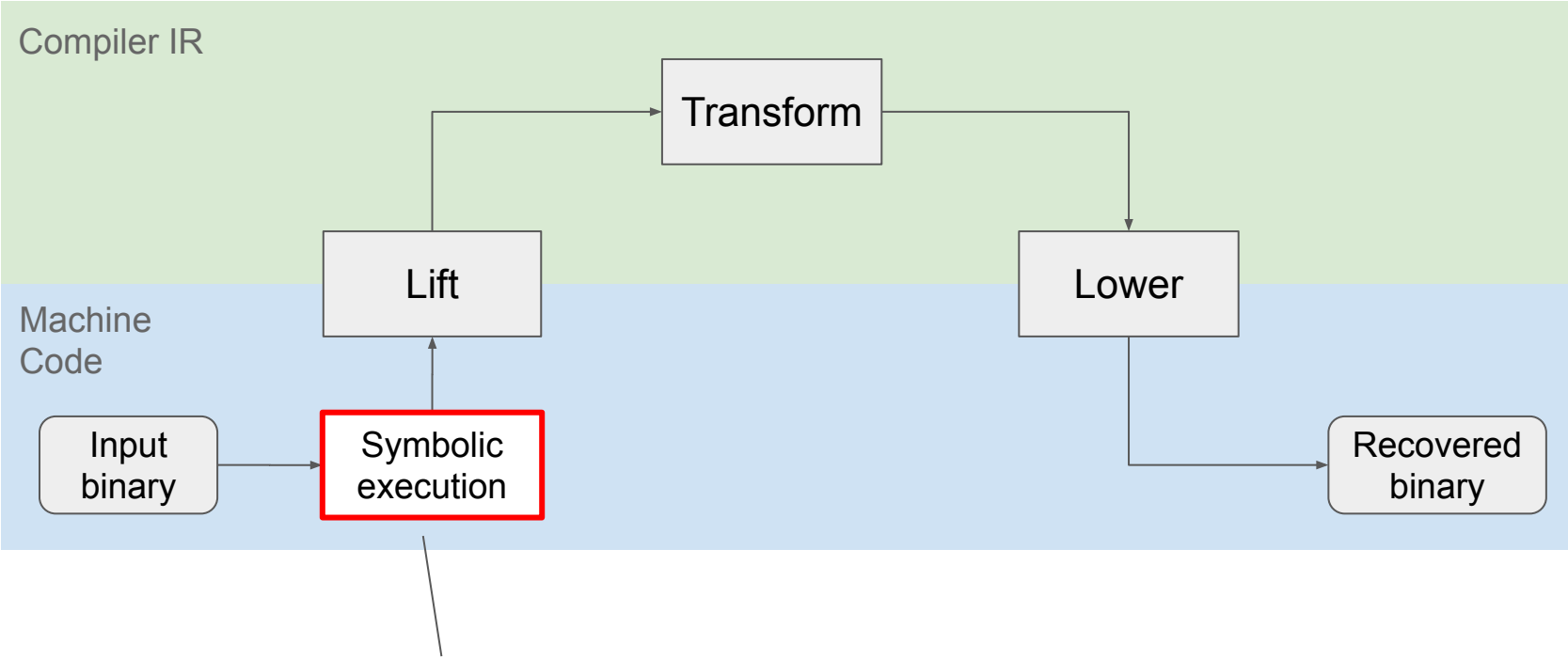


BinRec design



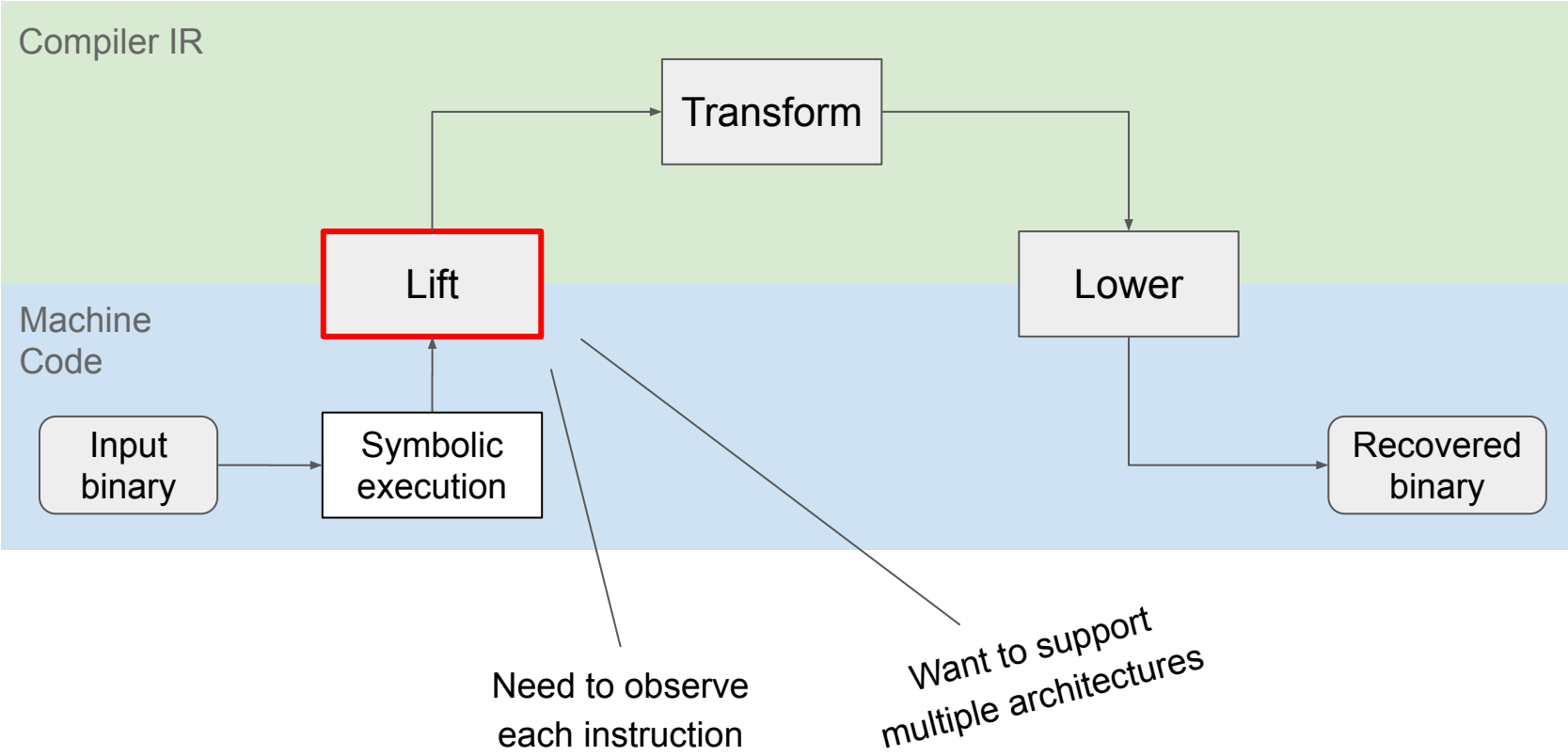
Sometimes we want more code coverage than a single code path

BinRec design

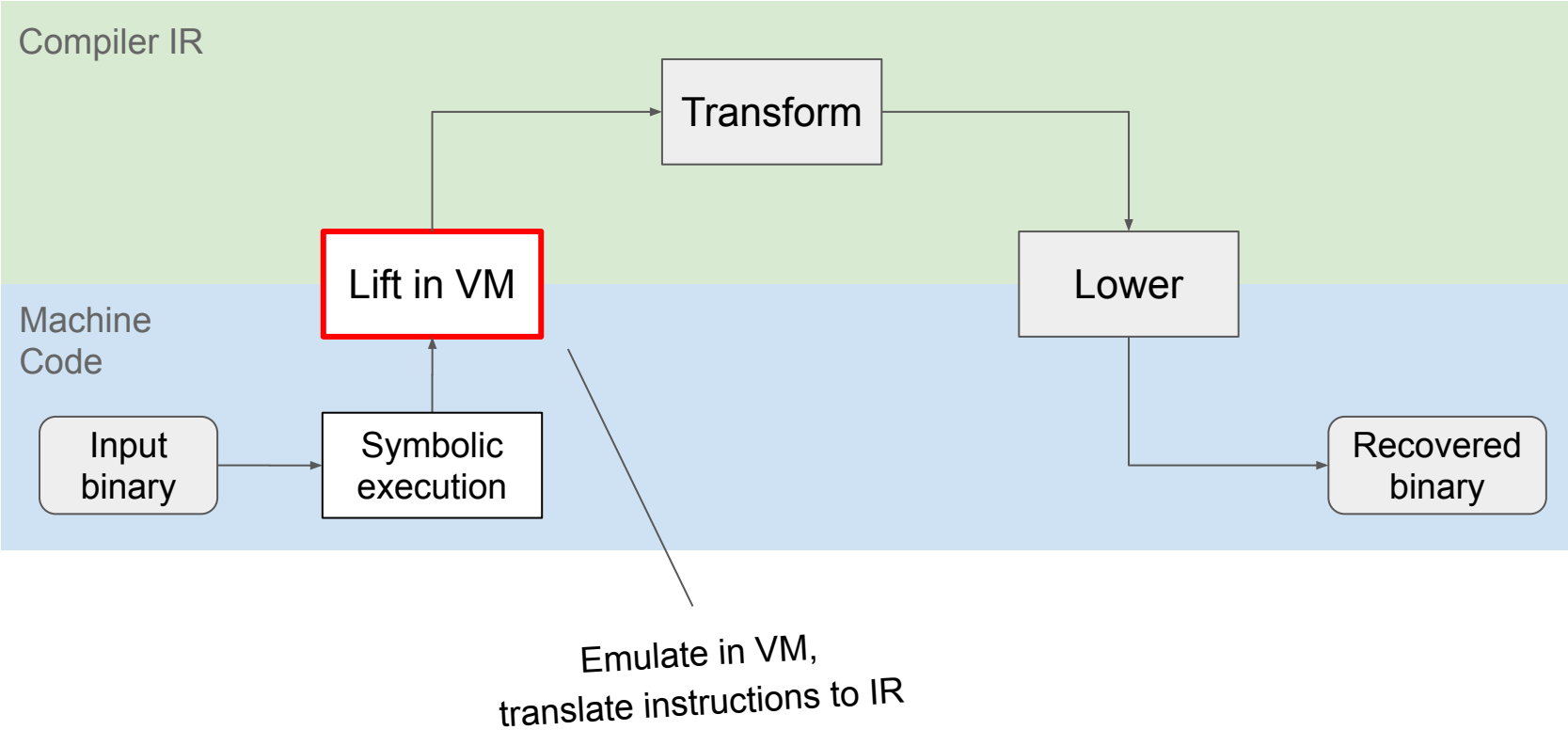


Run with “symbolic” input and follow both sides of a branch

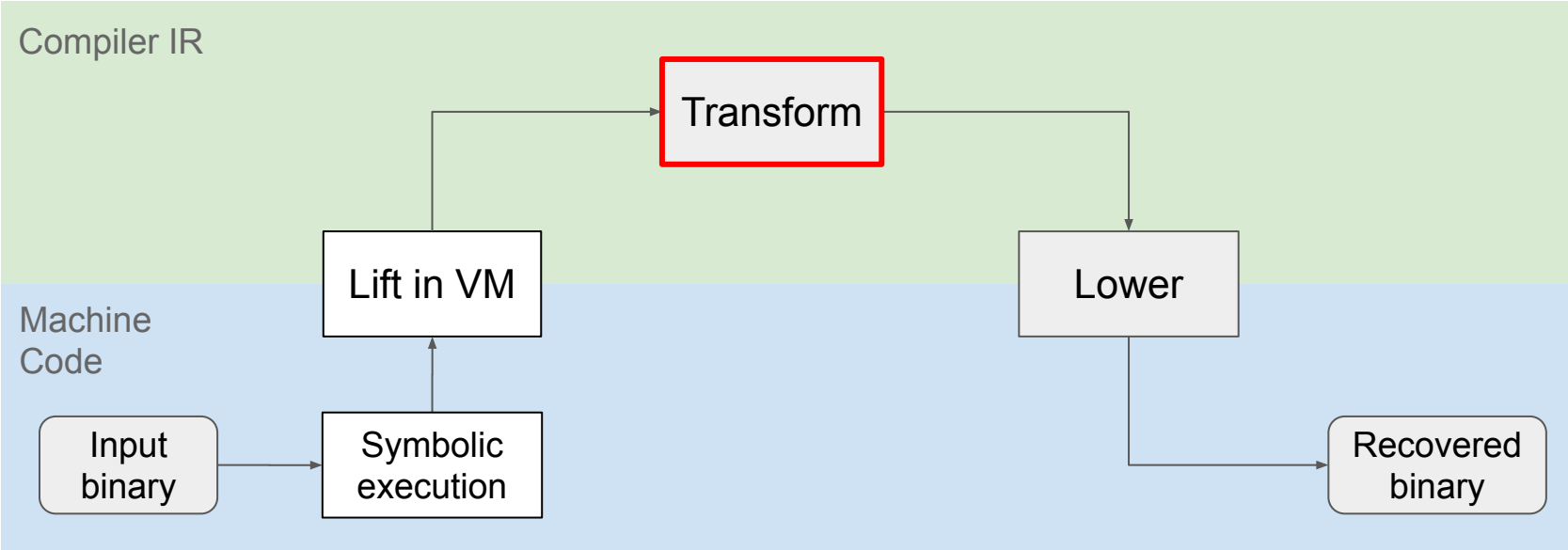
BinRec design



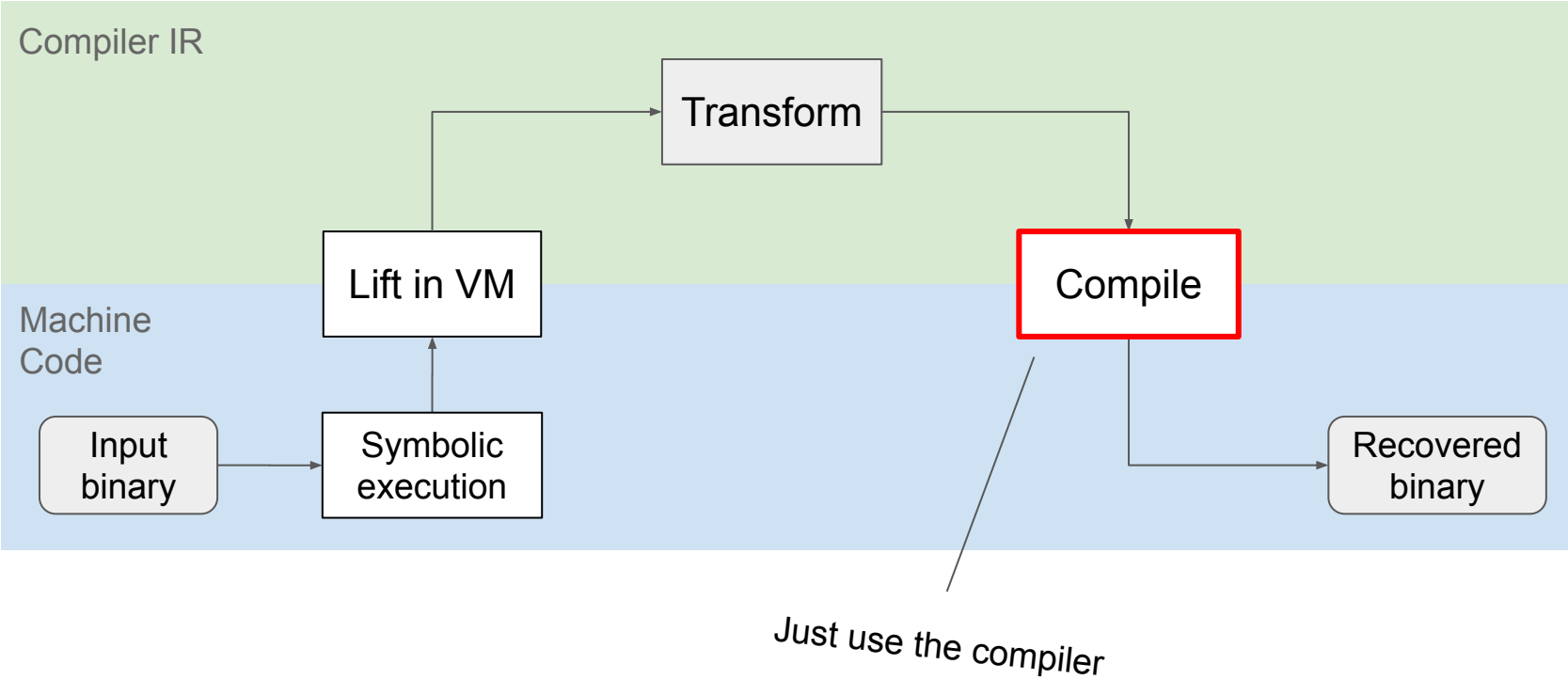
BinRec design



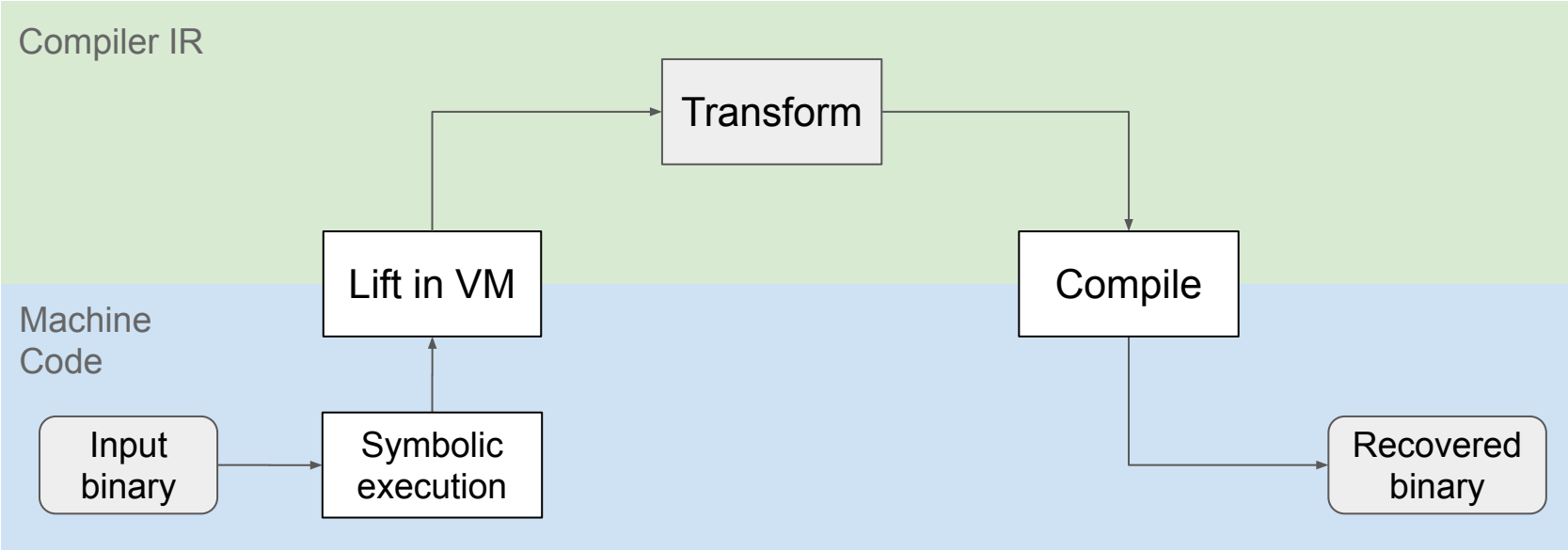
BinRec design



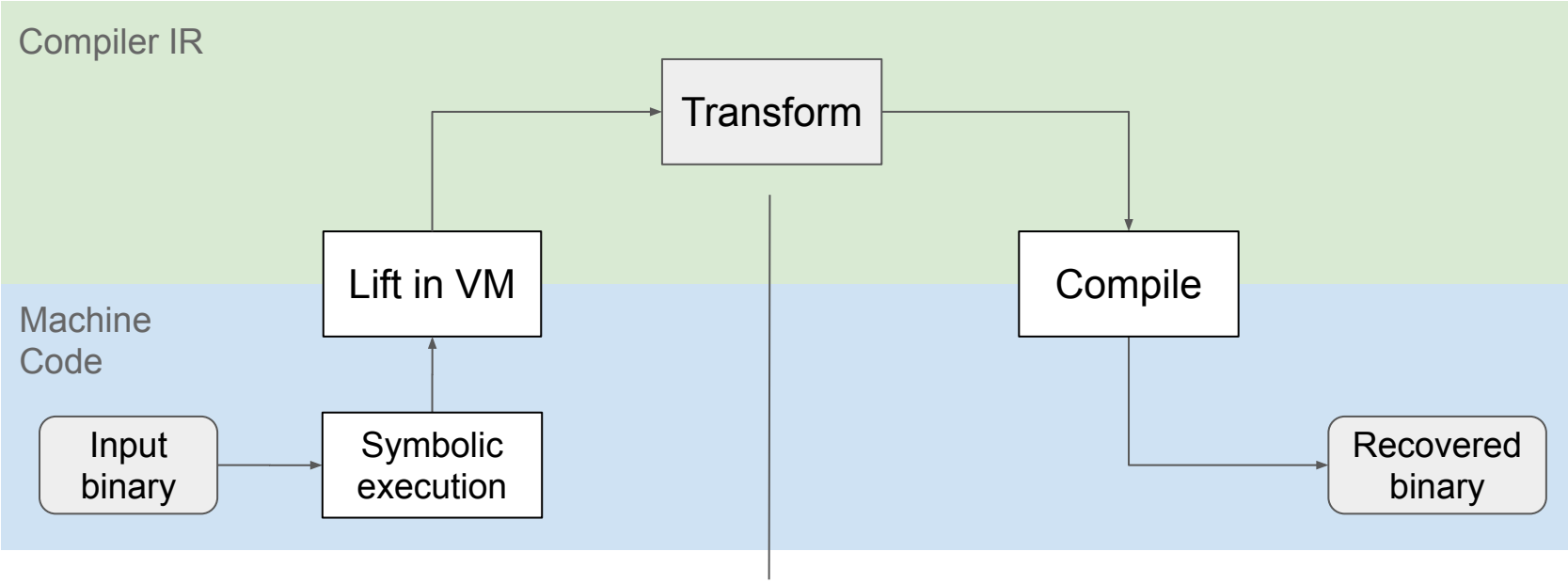
BinRec design



BinRec design



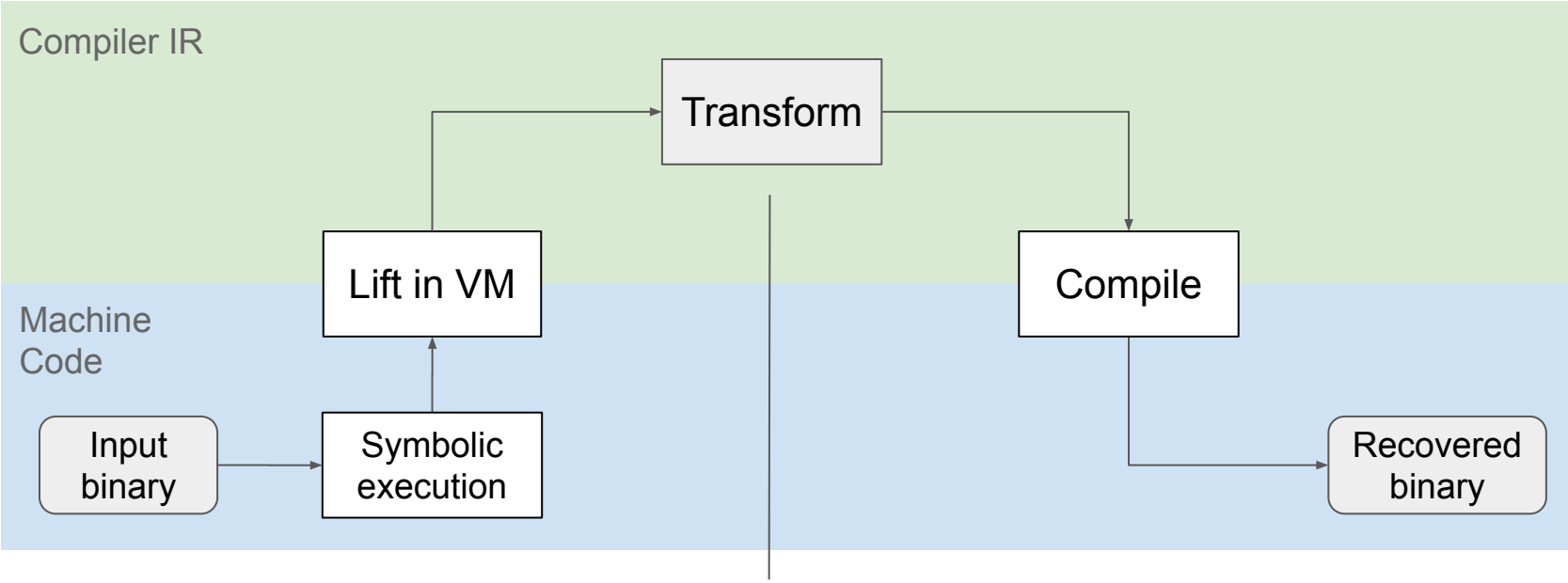
BinRec design



What about unlifted code paths?

```
...
if (getenv("SET_ME")) {
    puts("thanks!");  // not recovered!
}
...
```

BinRec design

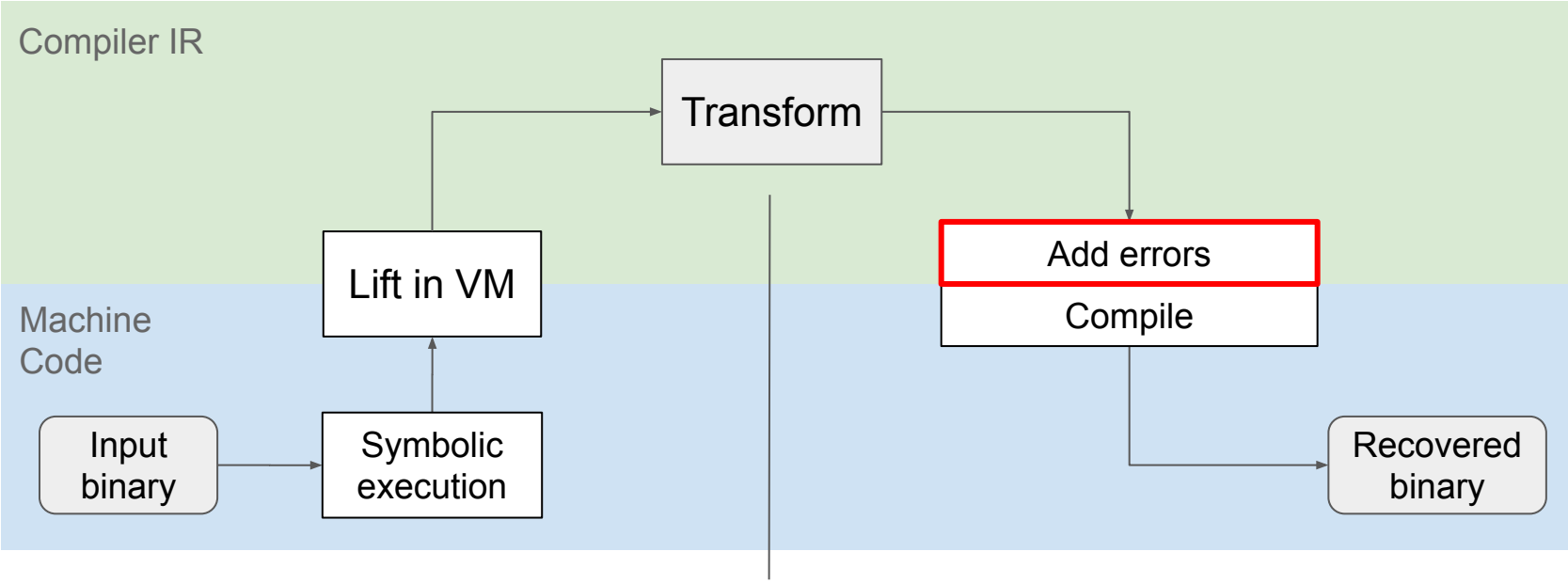


What about unlifted code paths?

1. do nothing (breaks conservative behavior)

...
getenv(“SET_ME”);
...

BinRec design

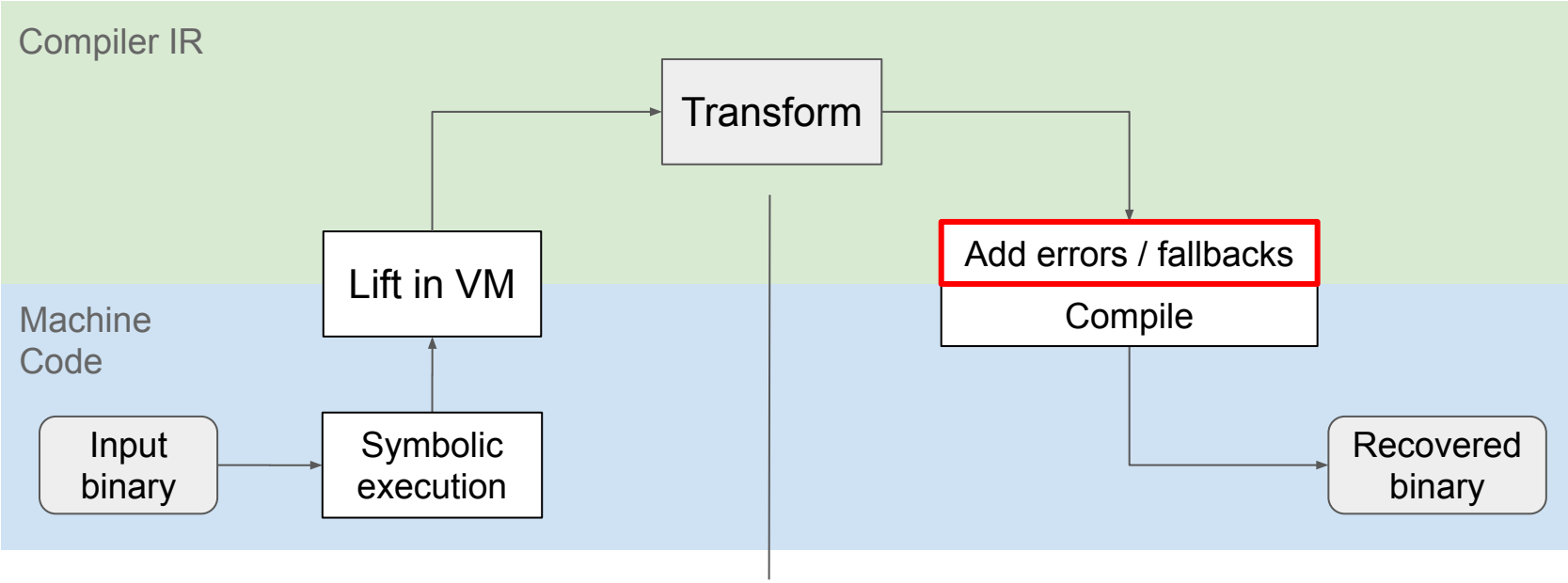


What about unlifted code paths?

2. yield error

```
...
if (getenv("SET_ME")) {
    abort();
}
...
```

BinRec design

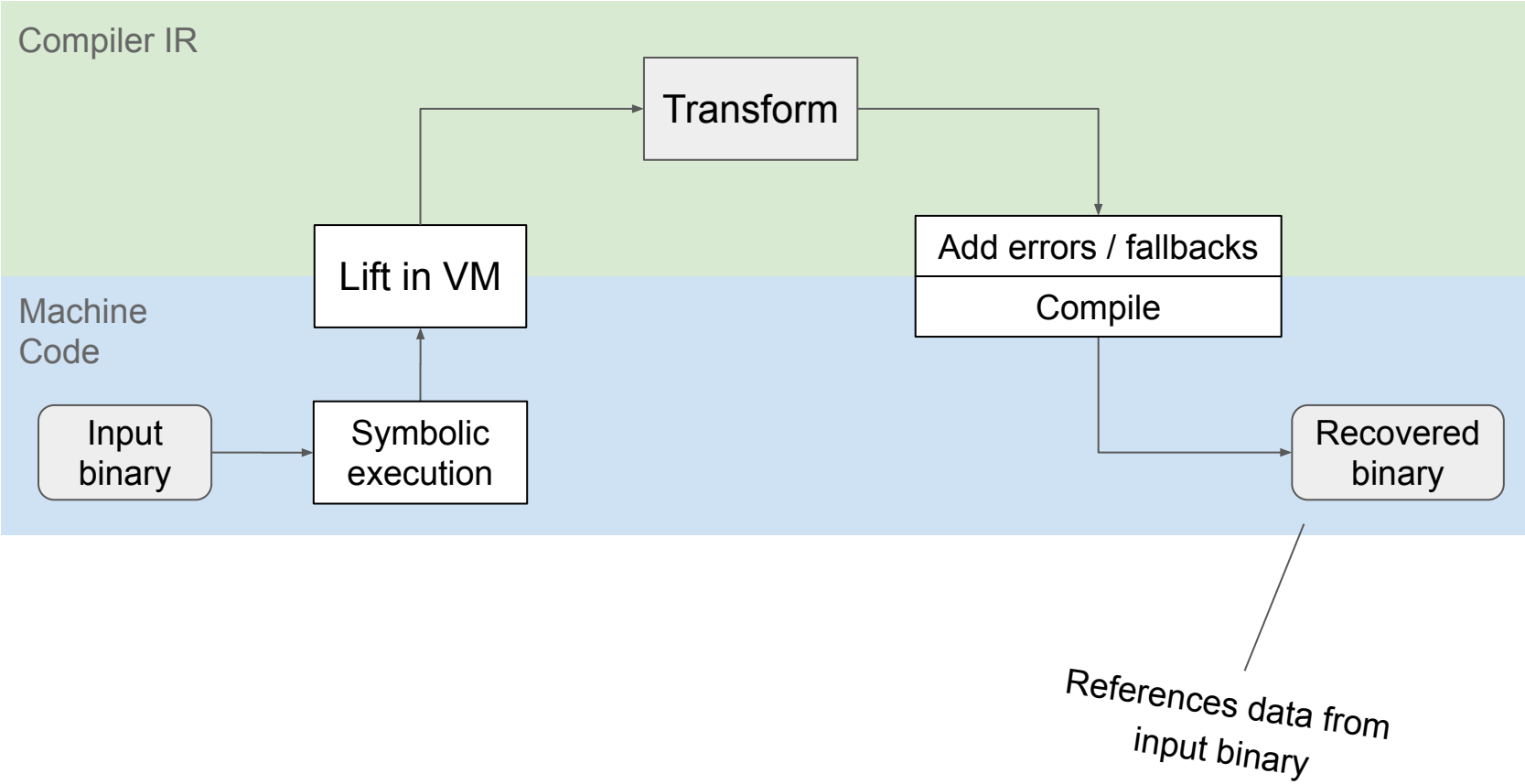


What about unlifted code paths?

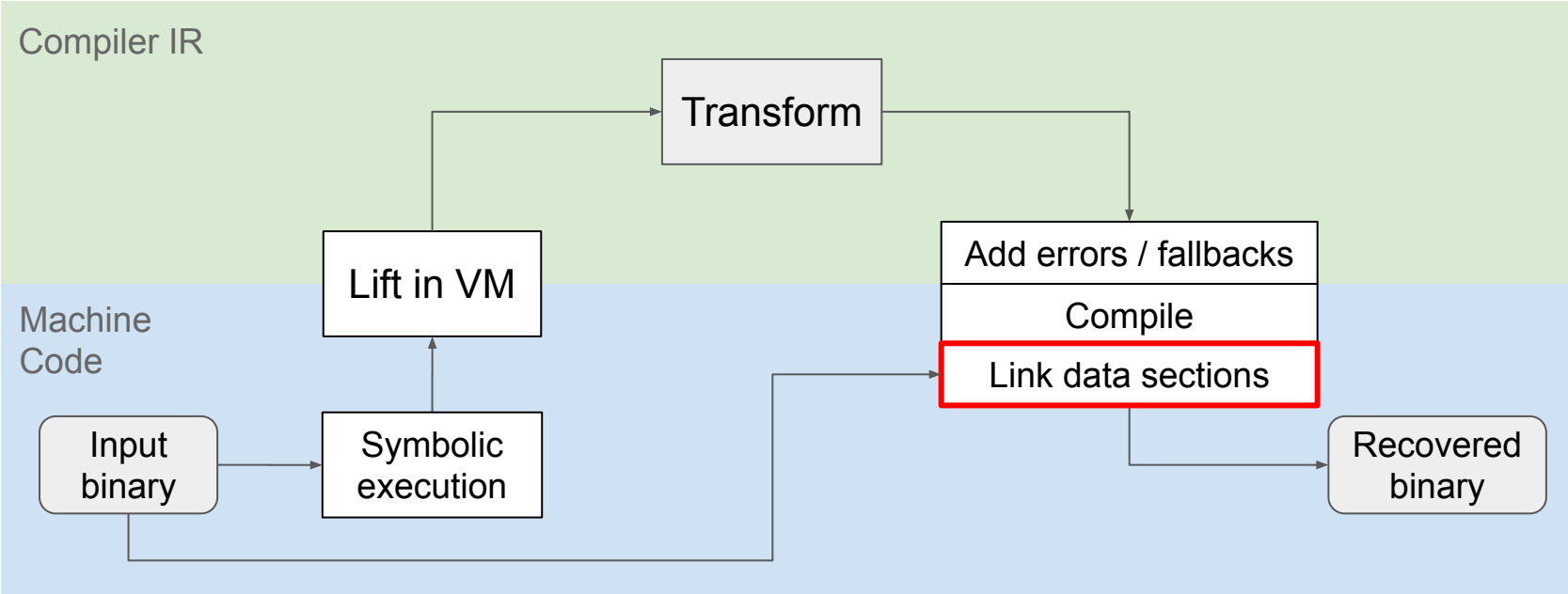
3. fallback to old code

```
...
if (getenv("SET_ME")) {
    goto old_code_address;
}
...
```

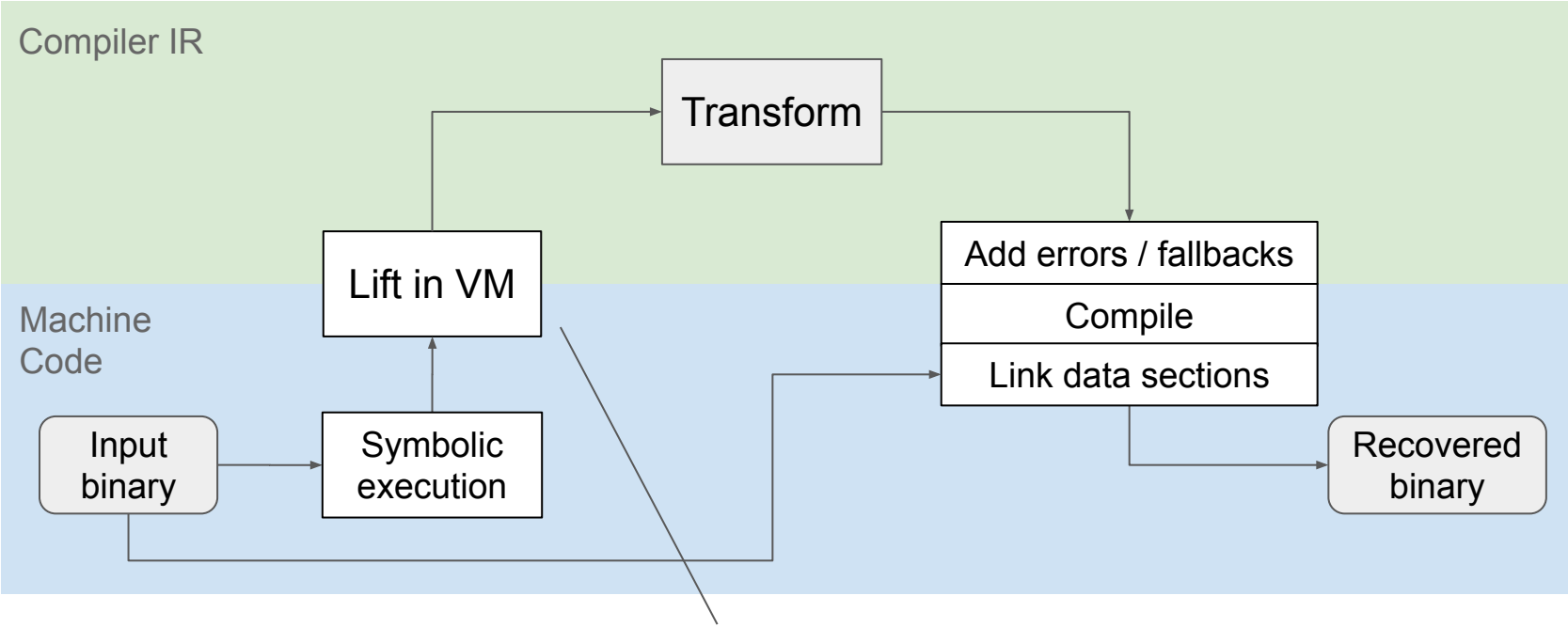
BinRec design



BinRec design

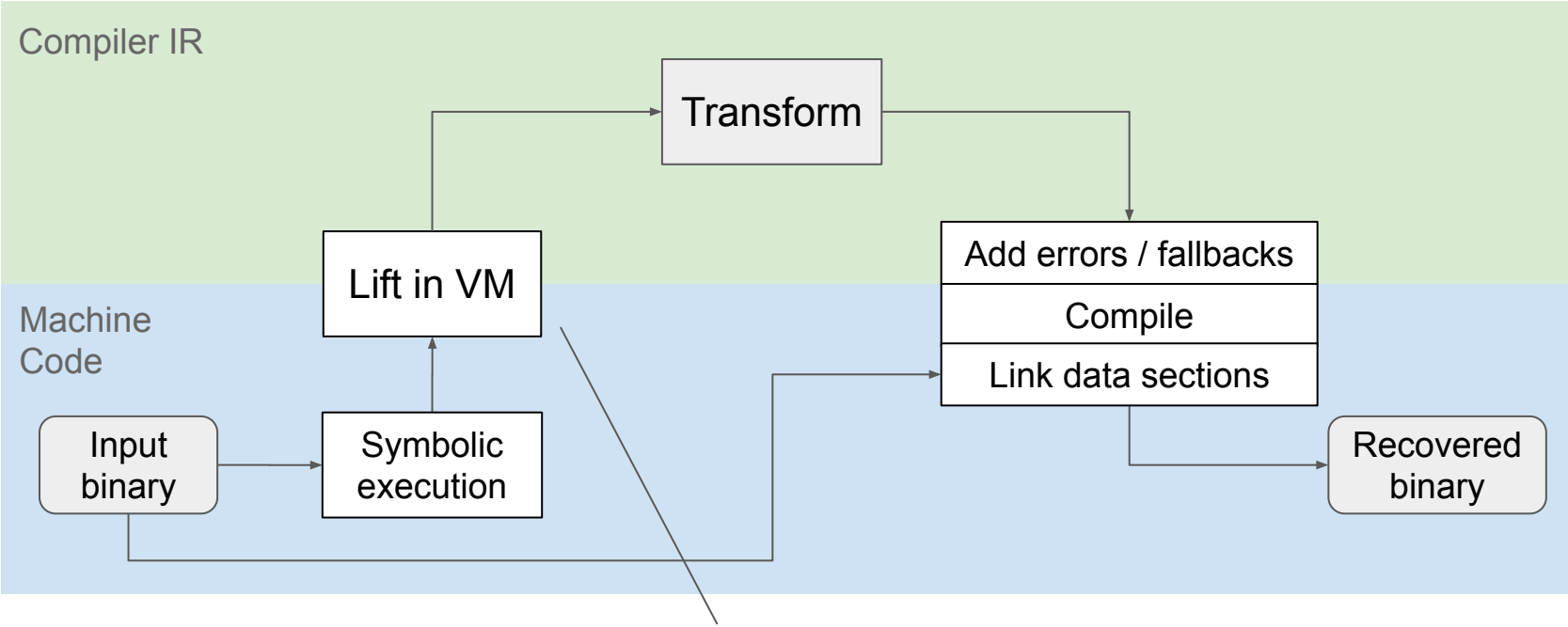


BinRec design



IR interacts with VM runtime

BinRec design

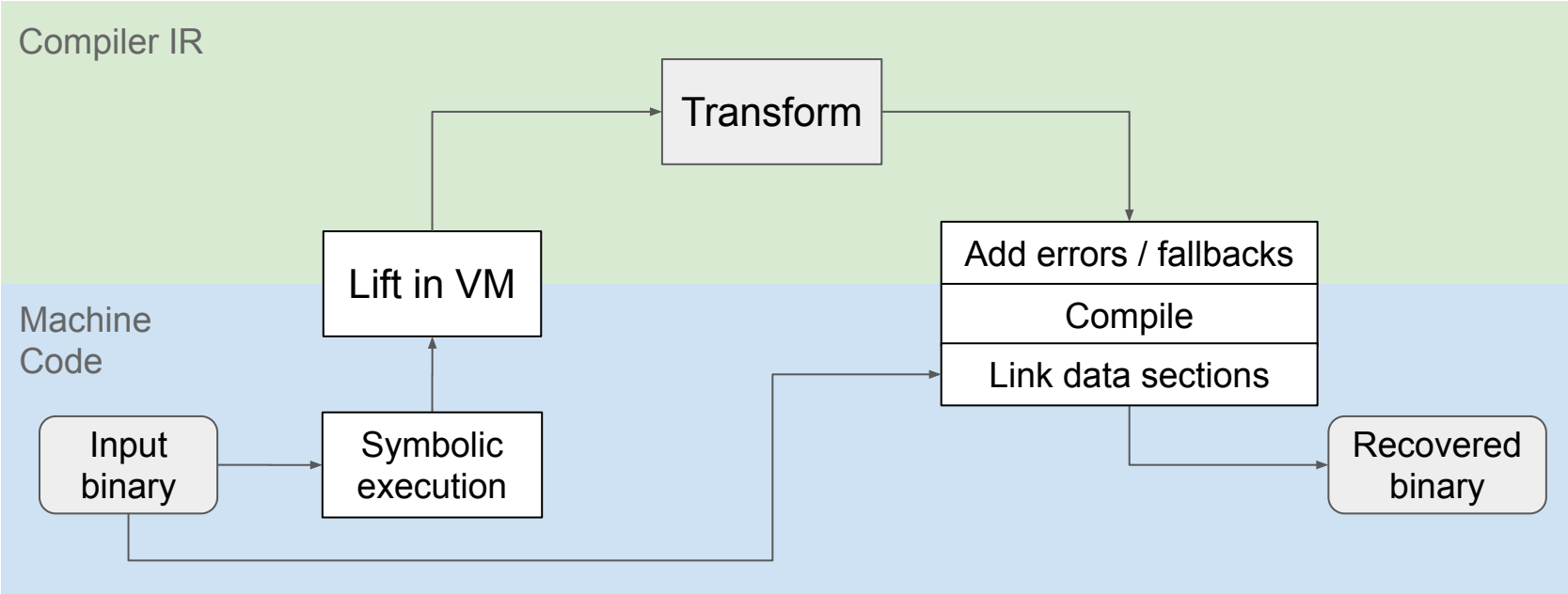


IR interacts with VM runtime

```
// machine code
0x1000:
add ebx, 1
jmp 0x1234
```

```
// Lifted code
emit_event(BASIC_BLOCK_START)
cpu_state.pc = 0x1000
ebx = &cpu_state.registers[R_EBX]
*ebx = *ebx + 1
cpu_state.icount++
cpu_state.pc = 0x1234
emit_event(BASIC_BLOCK_END)
```

BinRec design

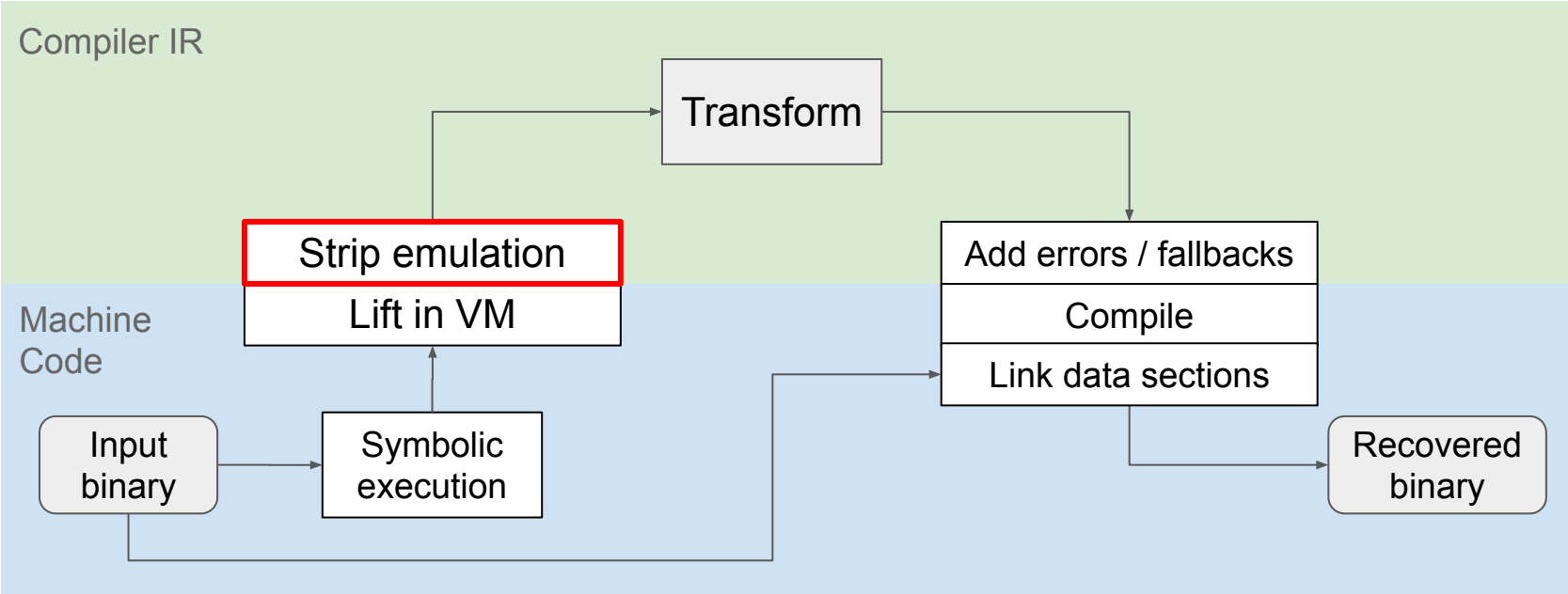



```
// machine code
0x1000:
add ebx, 1
jmp 0x1234
```

```
// Lifted code
emit_event(BASIC_BLOCK_START)
cpu_state.pc = 0x1000
ebx = &cpu_state.registers[R_EBX]
*ebx = *ebx + 1
cpu_state.icount++
cpu_state.pc = 0x1234
emit_event(BASIC_BLOCK_END)
```

events, counters

BinRec design



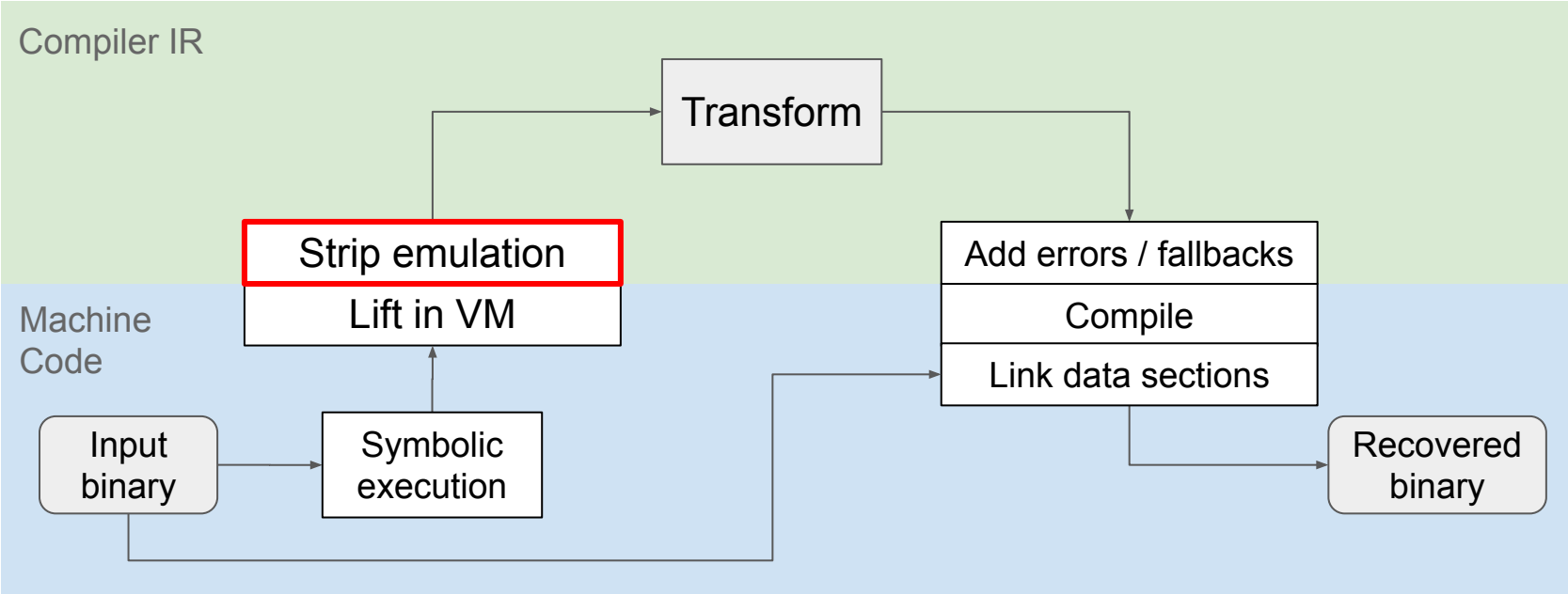

```
// machine code
0x1000:
add ebx, 1
jmp 0x1234
```

```
// Lifted code
emit_event(BASIC_BLOCK_START)
cpu_state.pc = 0x1000
ebx = &cpu_state.registers[R_EBX]
*ebx = *ebx + 1
cpu_state.icount++
cpu_state.pc = 0x1234
emit_event(BASIC_BLOCK_END)
```

control flow through virtual program counter

registers in CPU state

BinRec design

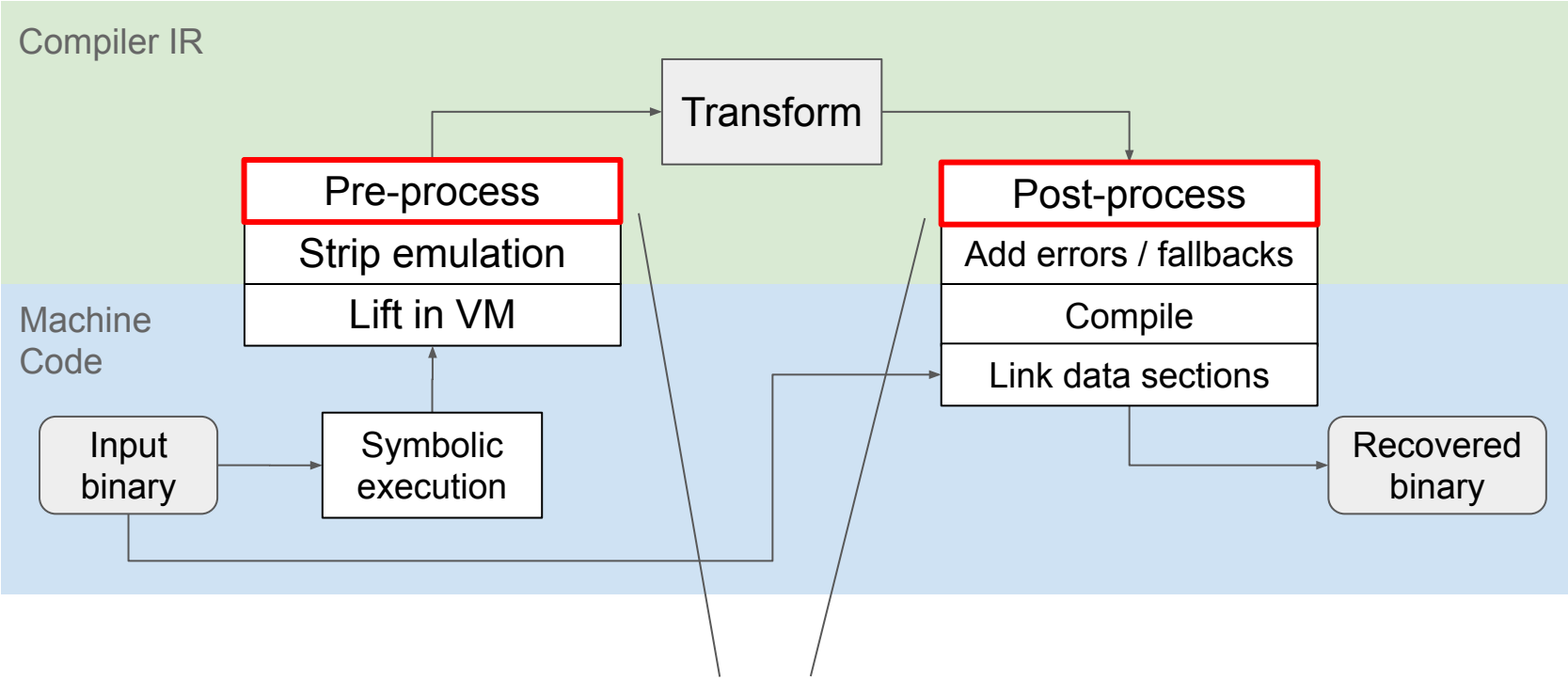



```
// machine code
0x1000:
add ebx, 1
jmp 0x1234
```

```
// Lifted code
emit_event(BASIC_BLOCK_START)
cpu_state.pc = 0x1000
ebx = &cpu_state.registers[R_EBX]
*ebx = *ebx + 1
cpu_state.icount++
cpu_state.pc = 0x1234
emit_event(BASIC_BLOCK_END)
```

```
// stripped code
global ebx
lifted_1000:
ebx = ebx + 1
goto lifted_1234
```

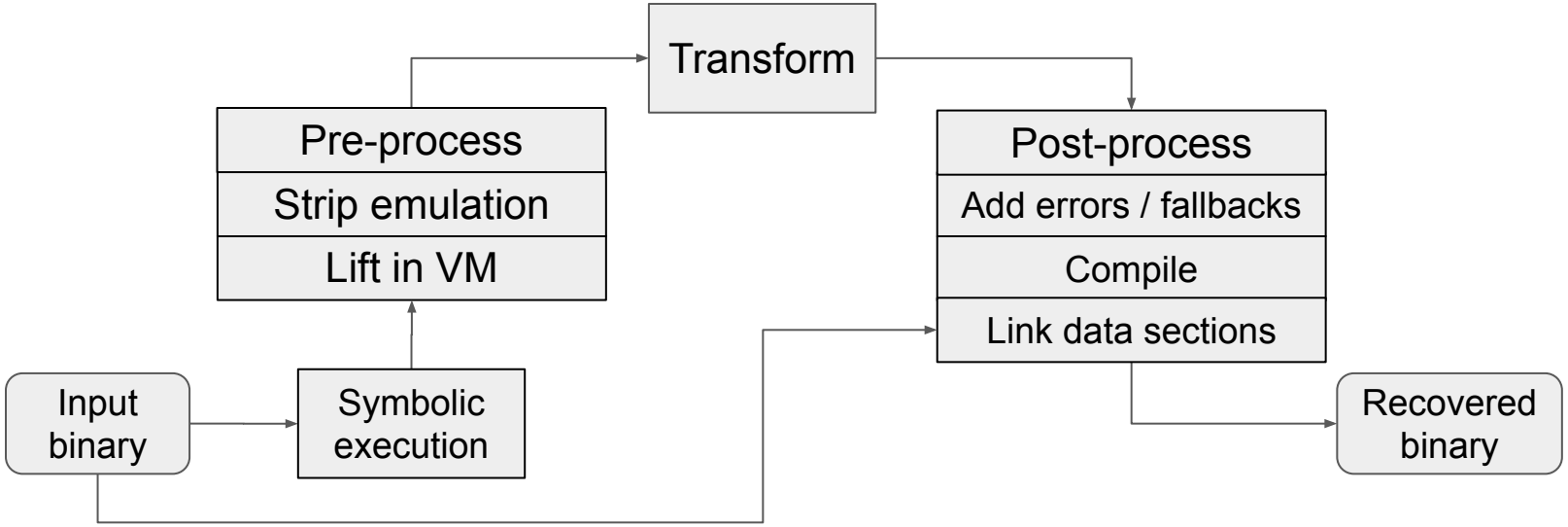
BinRec design



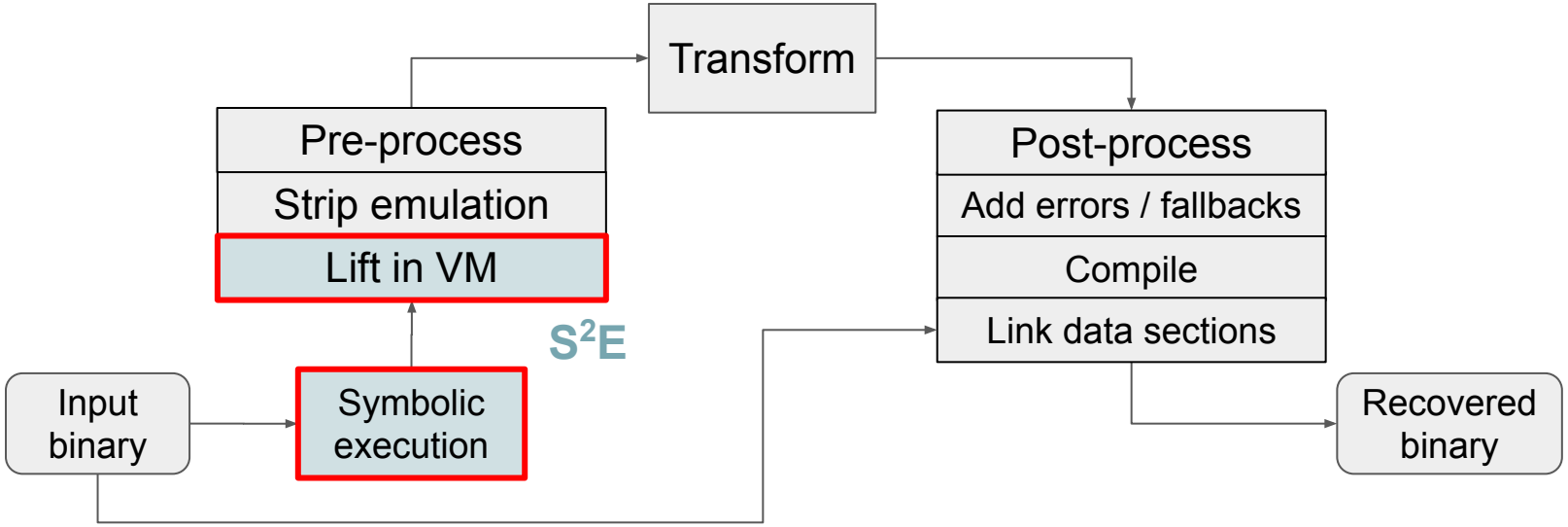
Needed to prevent over-optimization during transformations (details in paper)

This is quite bit of code

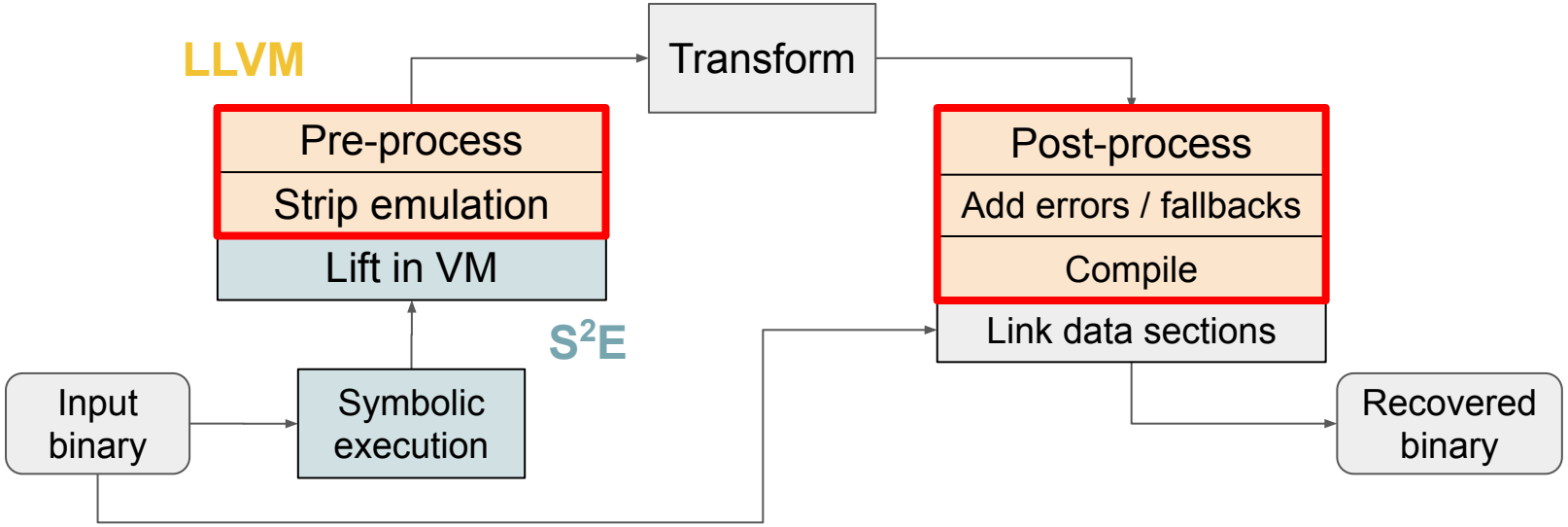
Implementation



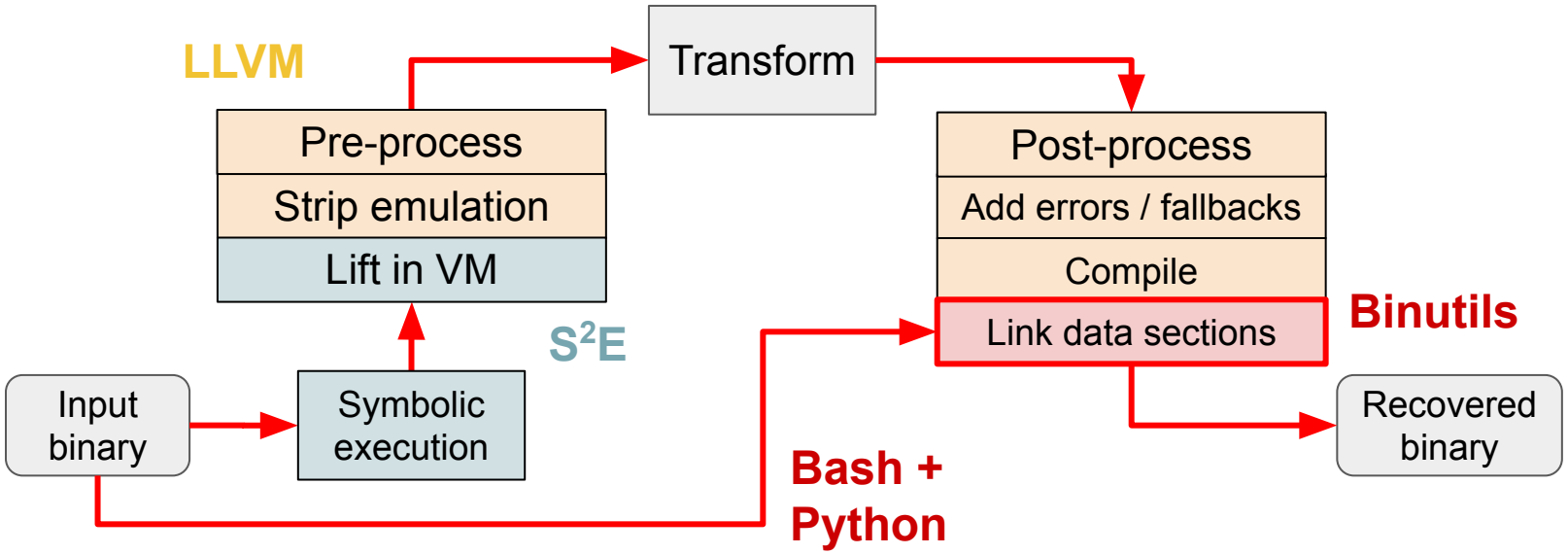
Implementation



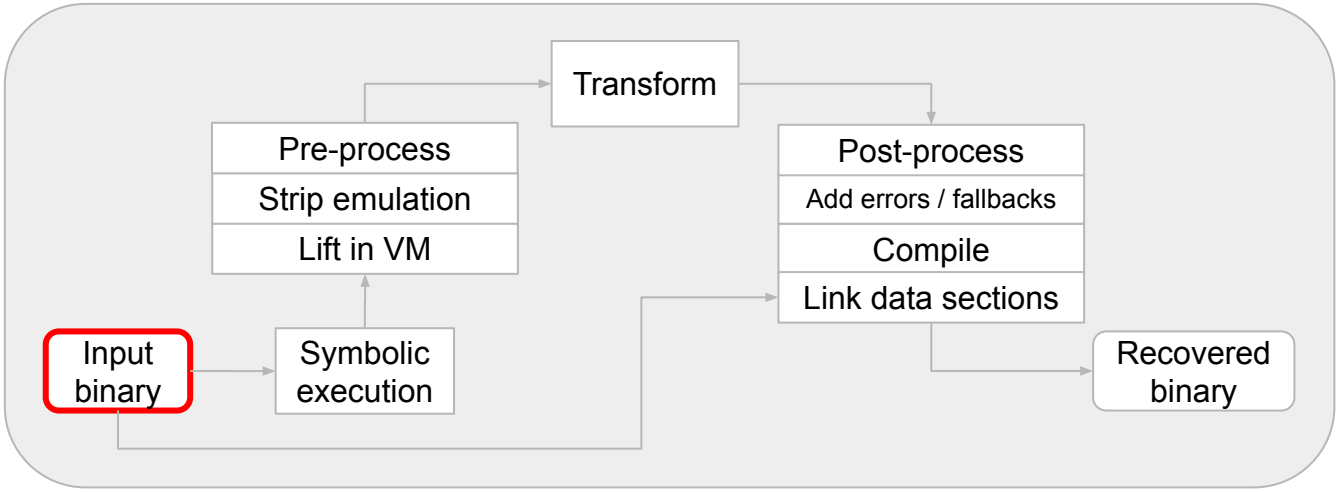
Implementation



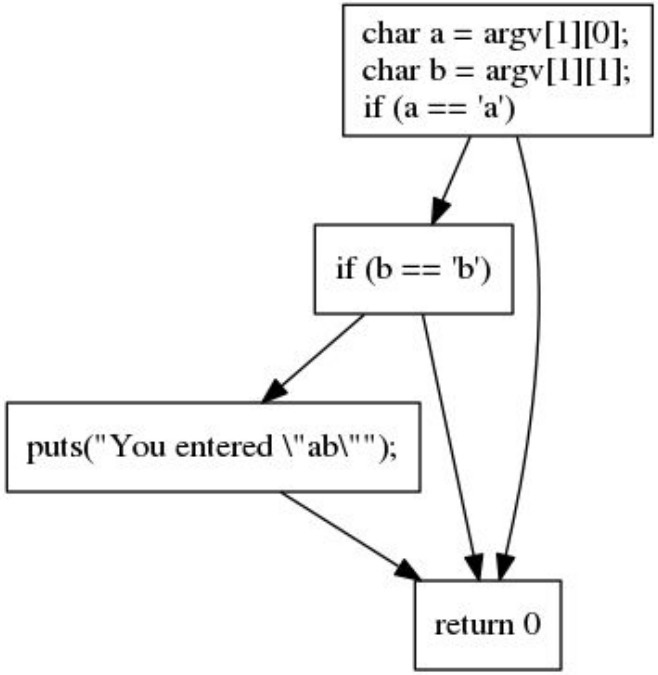
Implementation

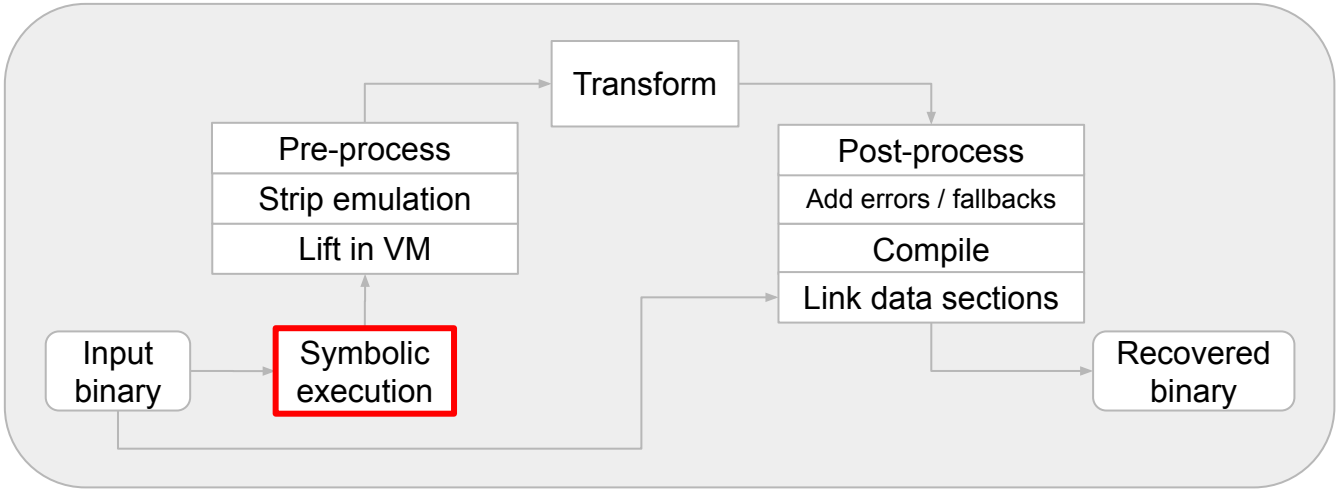


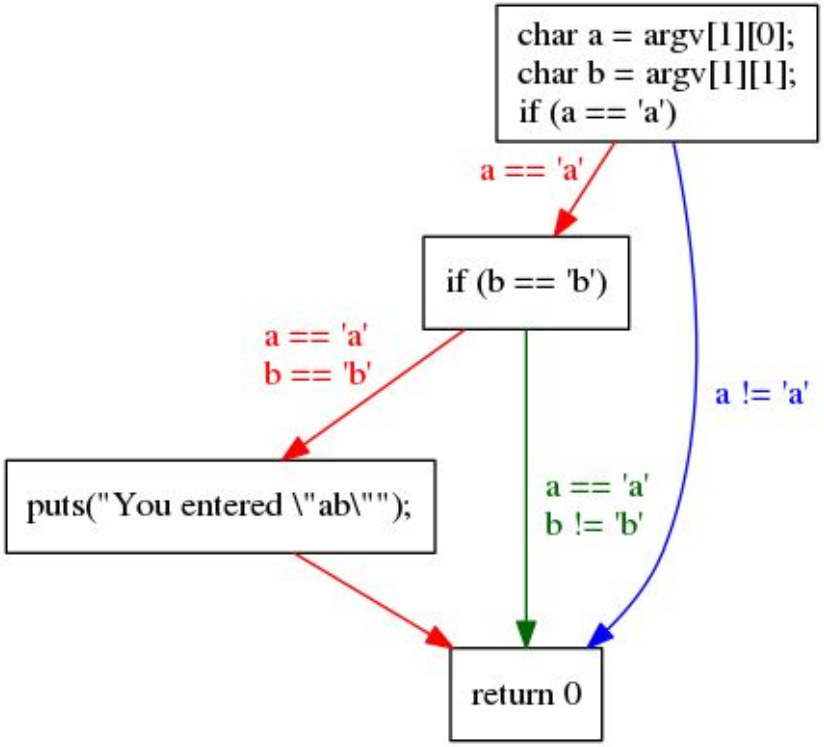
Case study

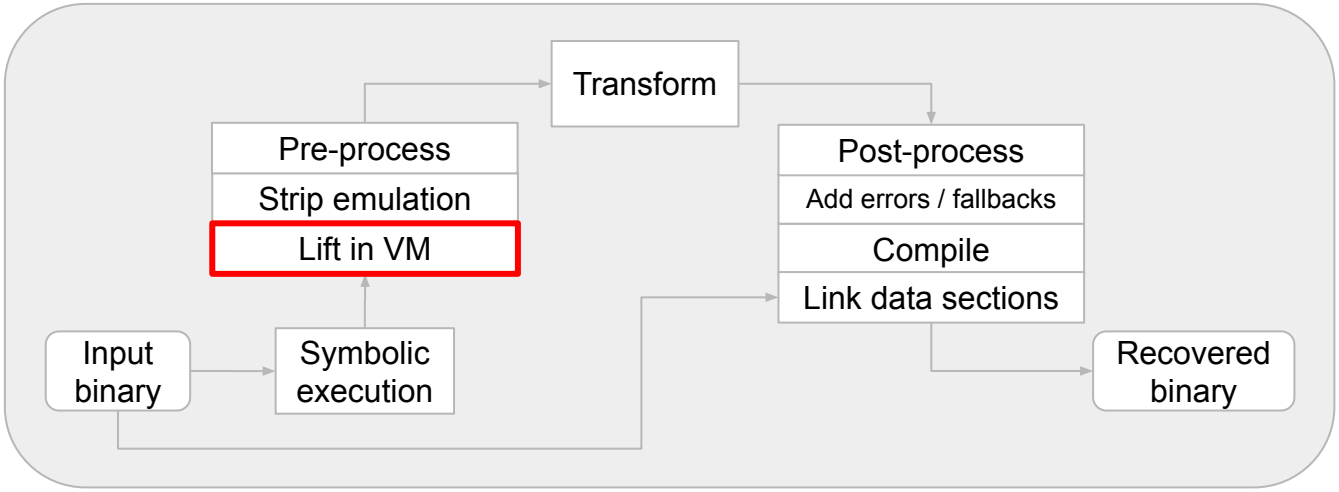


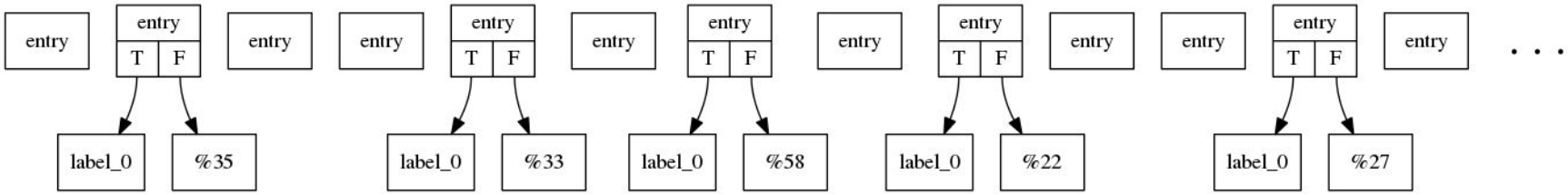

```
// ab.c
int main(int argc, char **argv) {
  char a = argv[1][0];
  char b = argv[1][1];
  if (a == 'a') {
    if (b == 'b') {
      puts("You entered \"ab\"");
    }
  }
  return 0;
}
```




Transform
Pre-process
Strip emulation
Lift in VM
Post-process
Add errors / fallbacks
Compile
Link data sections
Input binary
Symbolic execution
Recovered binary

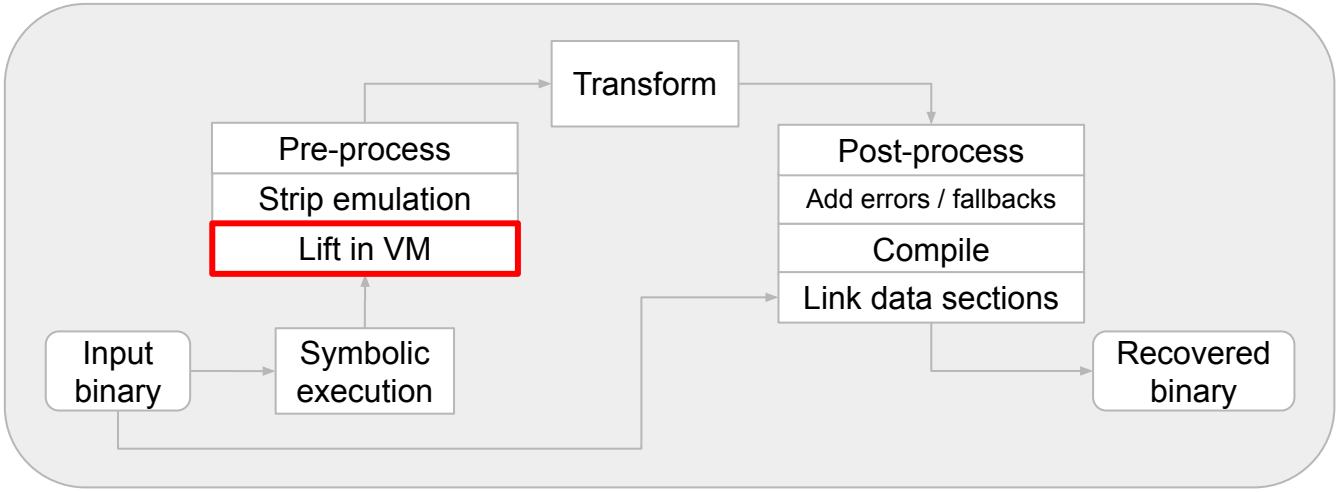

char a = argv[1][0];
char b = argv[1][1];
if (a == 'a')
a == 'a'
if (b == 'b')
a == 'a'
b == 'b'
a != 'a'
puts("You entered \"ab\"");
a == 'a'
b != 'b'
return 0





Raw code is heavily instrumented

- event triggers
- instruction counter
- program counter, registers, flags, etc. stored in CPU state in memory



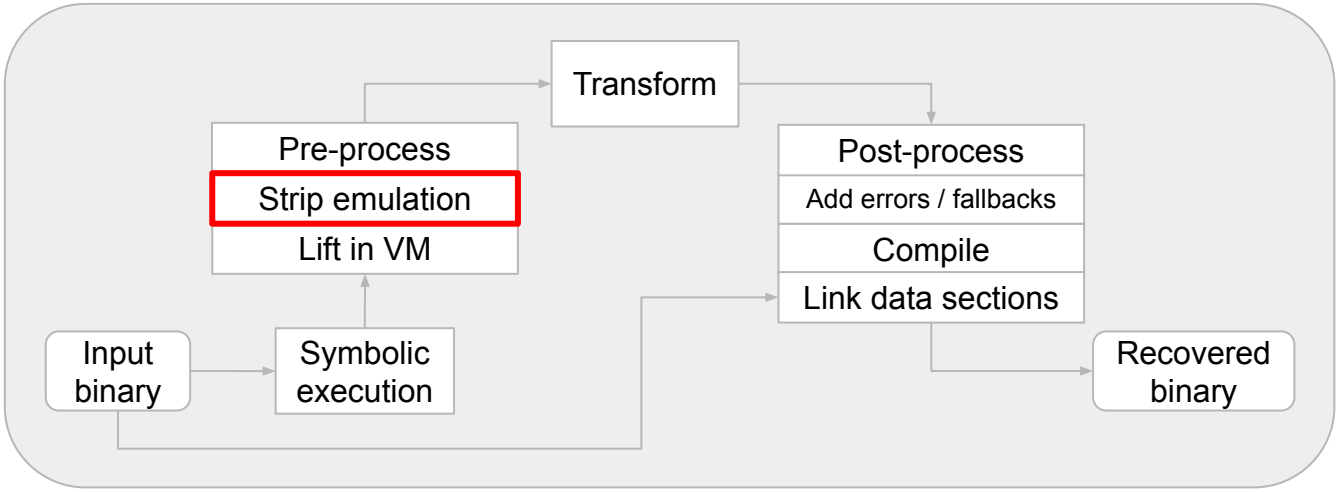
“heavily” instrumented

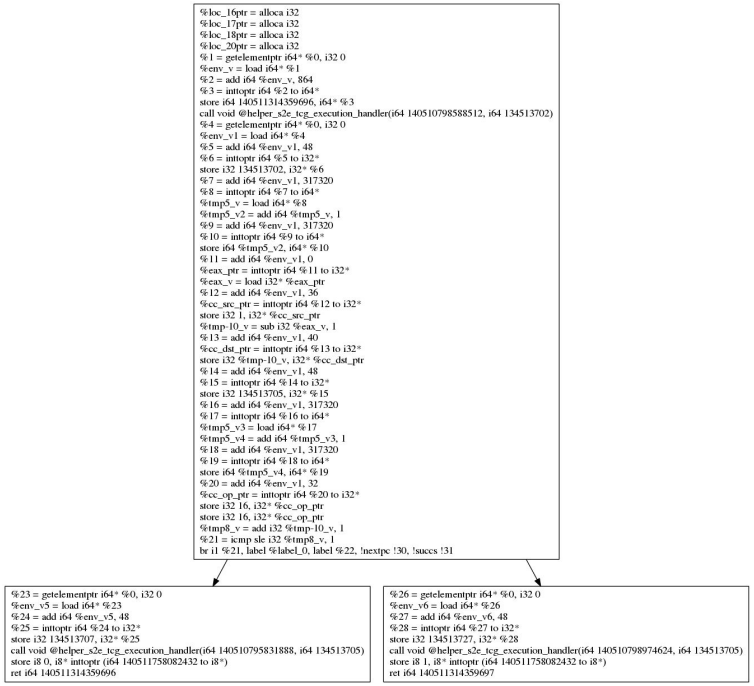
```
cmp eax, 1

jle label
```

```
%loc_16ptr = alloca i32
%loc_17ptr = alloca i32
%loc_18ptr = alloca i32
%loc_20ptr = alloca i32
%1 = getelementptr i64* %0, i32 0
%env_v = load i64* %1
%2 = add i64 %env_v, 864
%3 = inttoptr i64 %2 to i64*
store i64 140511314359696, i64* %3
call void @helper_s2e_tcg_execution_handler(i64 140510798588512, i64 134513702)
%4 = getelementptr i64* %0, i32 0
%env_v1 = load i64* %4
%5 = add i64 %env_v1, 48
%6 = inttoptr i64 %5 to i32*
store i32 134513702, i32* %6
%7 = add i64 %env_v1, 317320
%8 = inttoptr i64 %7 to i64*
%tmp5_v = load i64* %8
%tmp5_v2 = add i64 %tmp5_v, 1
%9 = add i64 %env_v1, 317320
%10 = inttoptr i64 %9 to i64*
store i64 %tmp5_v2, i64* %10
%11 = add i64 %env_v1, 0
%eax_ptr = inttoptr i64 %11 to i32*
%eax_v = load i32* %eax_ptr
%12 = add i64 %env_v1, 36
%cc_src_ptr = inttoptr i64 %12 to i32*
store i32 1, i32* %cc_src_ptr
%tmp-10_v = sub i32 %eax_v, 1
%13 = add i64 %env_v1, 40
%cc_dst_ptr = inttoptr i64 %13 to i32*
store i32 %tmp-10_v, i32* %cc_dst_ptr
%14 = add i64 %env_v1, 48
%15 = inttoptr i64 %14 to i32*
store i32 134513705, i32* %15
%16 = add i64 %env_v1, 317320
%17 = inttoptr i64 %16 to i64*
%tmp5_v3 = load i64* %17
%tmp5_v4 = add i64 %tmp5_v3, 1
%18 = add i64 %env_v1, 317320
%19 = inttoptr i64 %18 to i64*
store i64 %tmp5_v4, i64* %19
%20 = add i64 %env_v1, 32
%cc_op_ptr = inttoptr i64 %20 to i32*
store i32 16, i32* %cc_op_ptr
store i32 16, i32* %cc_op_ptr
%tmp8_v = add i32 %tmp-10_v, 1
%21 = icmp sle i32 %tmp8_v, 1
br i1 %21, label %label_0, label %22, !nextpc !30, !succs !31
```

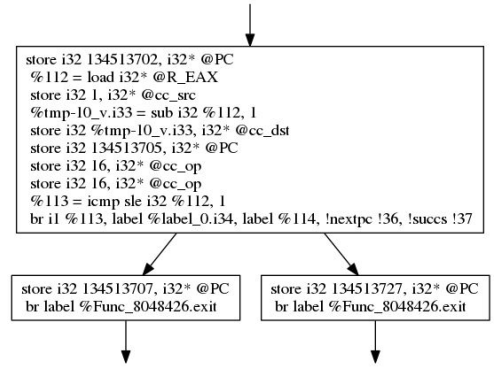
```
%23 = getelementptr i64* %0, i32 0
%env_v5 = load i64* %23
%24 = add i64 %env_v5, 48
%25 = inttoptr i64 %24 to i32*
store i32 134513707, i32* %25
call void @helper_s2e_tcg_execution_handler(i64 140510795831888, i64 134513705)
store i8 0, i8* inttoptr (i64 140511758082432 to i8*)
ret i64 140511314359696
```

```
%26 = getelementptr i64* %0, i32 0
%env_v6 = load i64* %26
%27 = add i64 %env_v6, 48
%28 = inttoptr i64 %27 to i32*
store i32 134513727, i32* %28
call void @helper_s2e_tcg_execution_handler(i64 140510798974624, i64 134513705)
store i8 1, i8* inttoptr (i64 140511758082432 to i8*)
ret i64 140511314359697
```

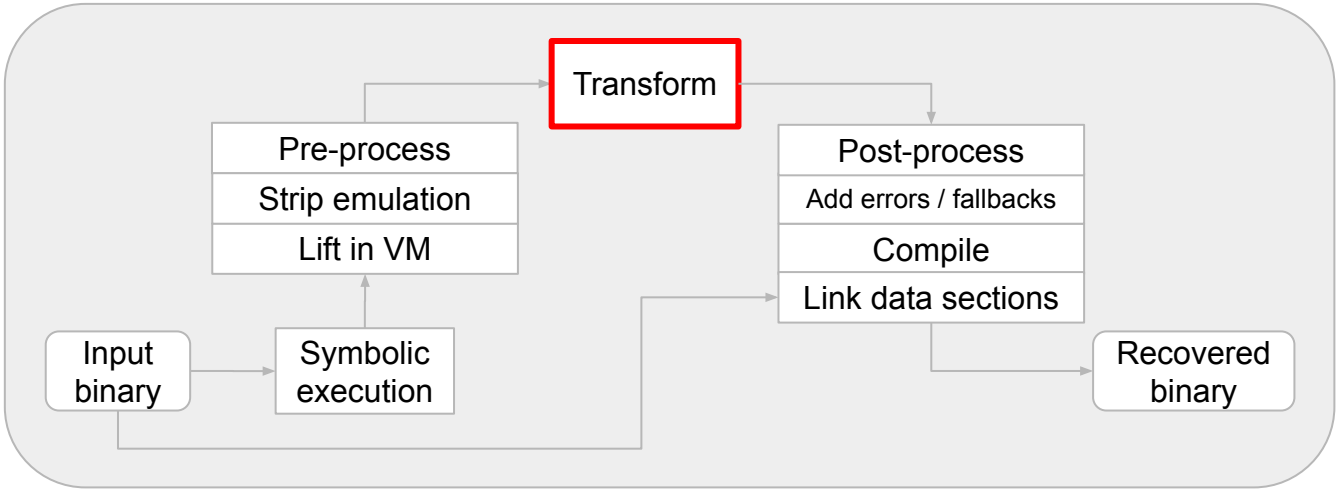

Transform
Pre-process
Strip emulation
Lift in VM
Post-process
Add errors / fallbacks
Compile
Link data sections
Input binary
Symbolic execution
Recovered binary

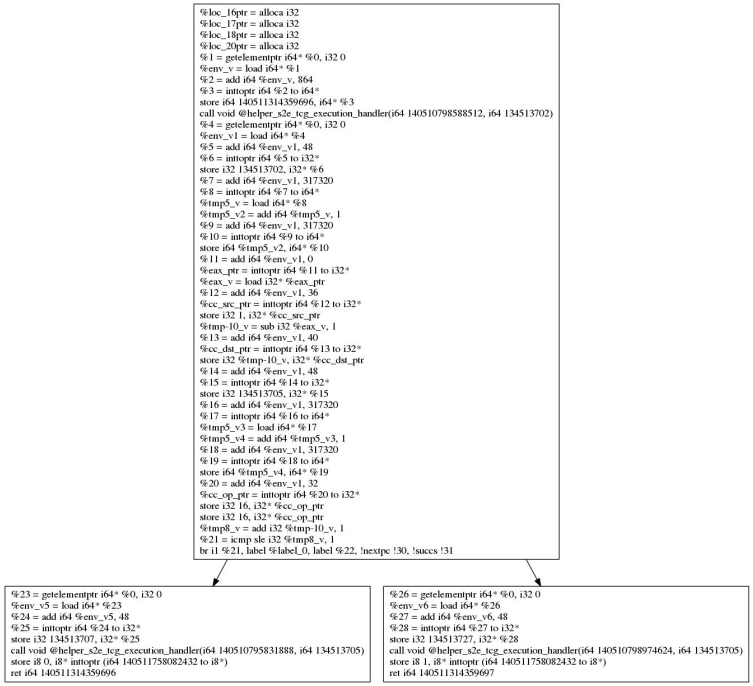

%loc_16ptr = alloca i32
%loc_17ptr = alloca i32
%loc_18ptr = alloca i32
%loc_20ptr = alloca i32
%1 = getelementptr i64* %0, i32 0
%env_v = load i64* %1
%2 = add i64 %env_v, 864
%3 = inttoptr i64 %2 to i64*
store i64 140511314359696, i64* %3
call void @helper_s2e_tcg_execution_handler(i64 140510798588512, i64 134513702)
%4 = getelementptr i64* %0, i32 0
%env_v1 = load i64* %4
%5 = add i64 %env_v1, 48
%6 = inttoptr i64 %5 to i32*
store i32 134513702, i32* %6
%7 = add i64 %env_v1, 317320
%8 = inttoptr i64 %7 to i64*
%tmp5_v = load i64* %8
%tmp5_v2 = add i64 %tmp5_v, 1
%9 = add i64 %env_v1, 317320
%10 = inttoptr i64 %9 to i64*
store i64 %tmp5_v2, i64* %10
%11 = add i64 %env_v1, 0
%eax_ptr = inttoptr i64 %11 to i32*
%eax_v = load i32* %eax_ptr
%12 = add i64 %env_v1, 36
%cc_src_ptr = inttoptr i64 %12 to i32*
store i32 1, i32* %cc_src_ptr
%tmp-10_v = sub i32 %eax_v, 1
%13 = add i64 %env_v1, 40
%cc_dst_ptr = inttoptr i64 %13 to i32*
store i32 %tmp-10_v, i32* %cc_dst_ptr
%14 = add i64 %env_v1, 48
%15 = inttoptr i64 %14 to i32*
store i32 134513705, i32* %15
%16 = add i64 %env_v1, 317320
%17 = inttoptr i64 %16 to i64*
%tmp5_v3 = load i64* %17
%tmp5_v4 = add i64 %tmp5_v3, 1
%18 = add i64 %env_v1, 317320
%19 = inttoptr i64 %18 to i64*
store i64 %tmp5_v4, i64* %19
%20 = add i64 %env_v1, 32
%cc_op_ptr = inttoptr i64 %20 to i32*
store i32 16, i32* %cc_op_ptr
store i32 16, i32* %cc_op_ptr
%tmp8_v = add i32 %tmp-10_v, 1
%21 = icmp sle i32 %tmp8_v, 1
br i1 %21, label %label_0, label %22, !nextpc !30, !succs !31
%23 = getelementptr i64* %0, i32 0
%env_v5 = load i64* %23
%24 = add i64 %env_v5, 48
%25 = inttoptr i64 %24 to i32*
store i32 134513707, i32* %25
call void @helper_s2e_tcg_execution_handler(i64 140510795831888, i64 134513705)
store i8 0, i8* inttoptr (i64 140511758082432 to i8*)
ret i64 140511314359696
%26 = getelementptr i64* %0, i32 0
%env_v6 = load i64* %26
%27 = add i64 %env_v6, 48
%28 = inttoptr i64 %27 to i32*
store i32 134513727, i32* %28
call void @helper_s2e_tcg_execution_handler(i64 140510798974624, i64 134513705)
store i8 1, i8* inttoptr (i64 140511758082432 to i8*)
ret i64 140511314359697

Raw

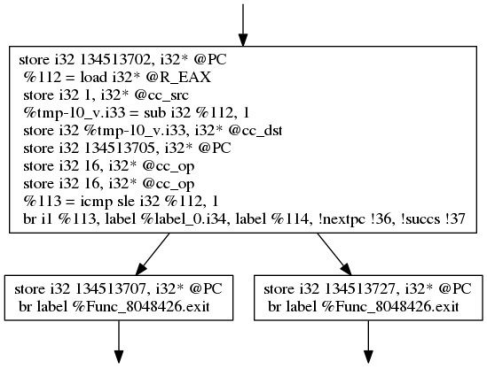

store i32 134513702, i32* @PC
%112 = load i32* @R_EAX
store i32 1, i32* @cc_src
%tmp-10_v.i33 = sub i32 %112, 1
store i32 %tmp-10_v.i33, i32* @cc_dst
store i32 134513705, i32* @PC
store i32 16, i32* @cc_op
store i32 16, i32* @cc_op
%113 = icmp sle i32 %112, 1
br i1 %113, label %label_0.i34, label %114, !nextpc !36, !succs !37
store i32 134513707, i32* @PC
br label %Func_8048426.exit
store i32 134513727, i32* @PC
br label %Func_8048426.exit

Pruned

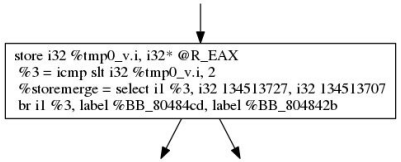




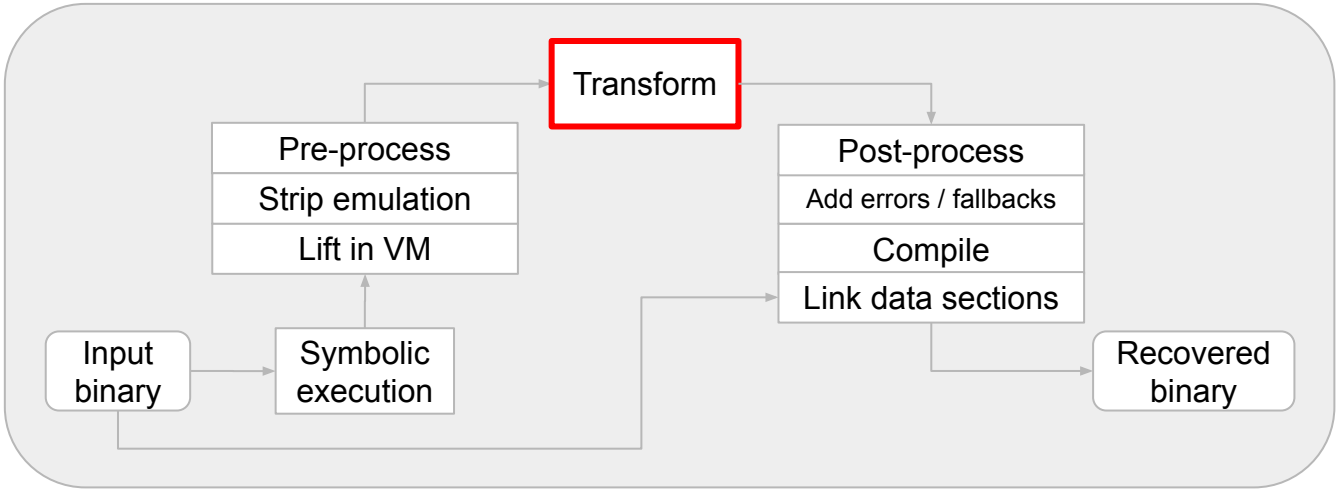
Raw

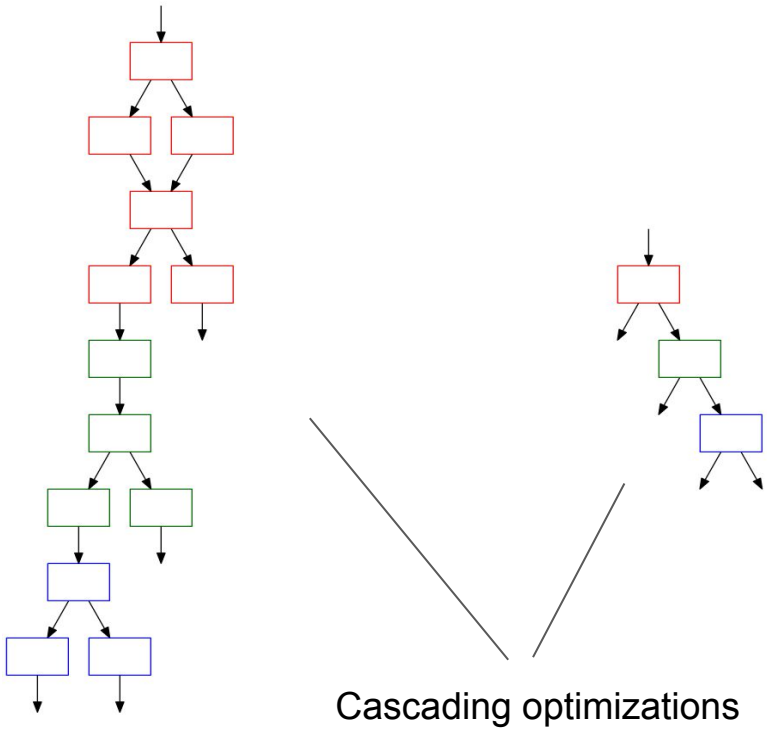


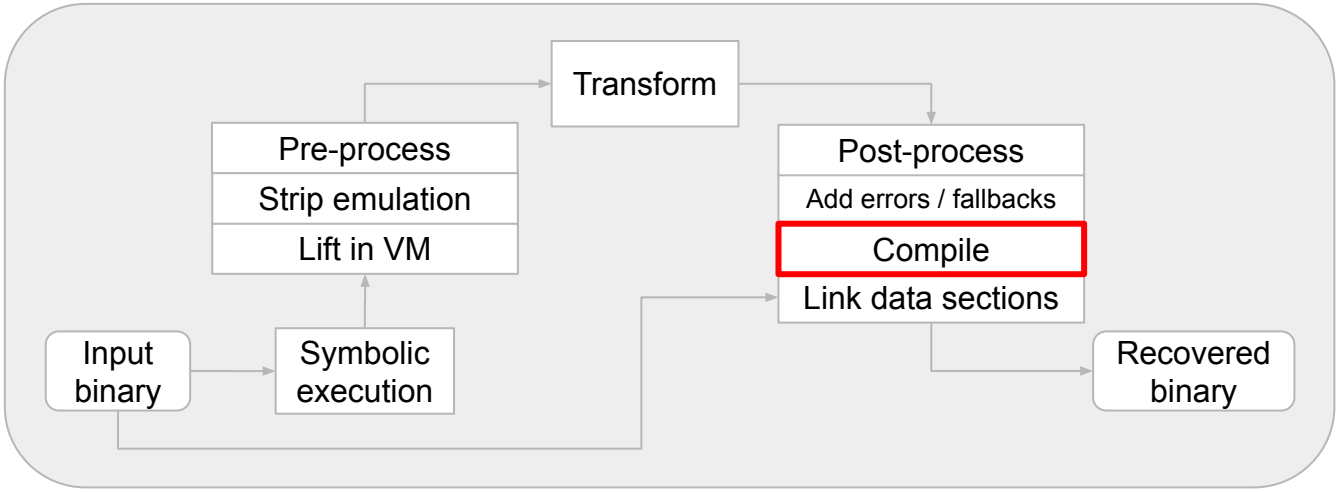
Pruned



Optimized

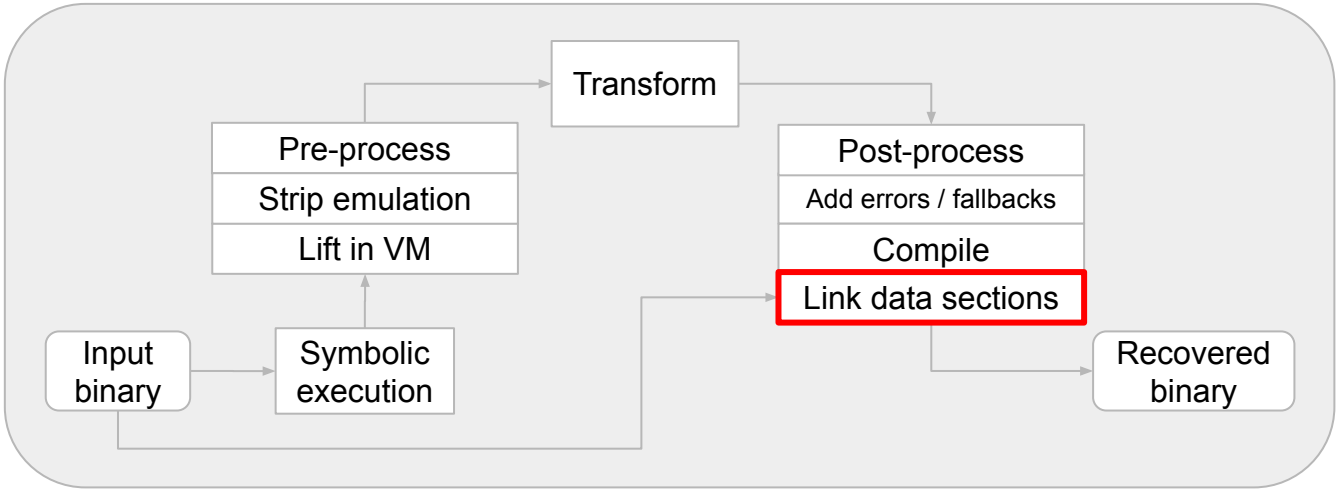

Transform
Pre-process
Strip emulation
Lift in VM
Post-process
Add errors / fallbacks
Compile
Link data sections
Input binary
Symbolic execution
Recovered binary

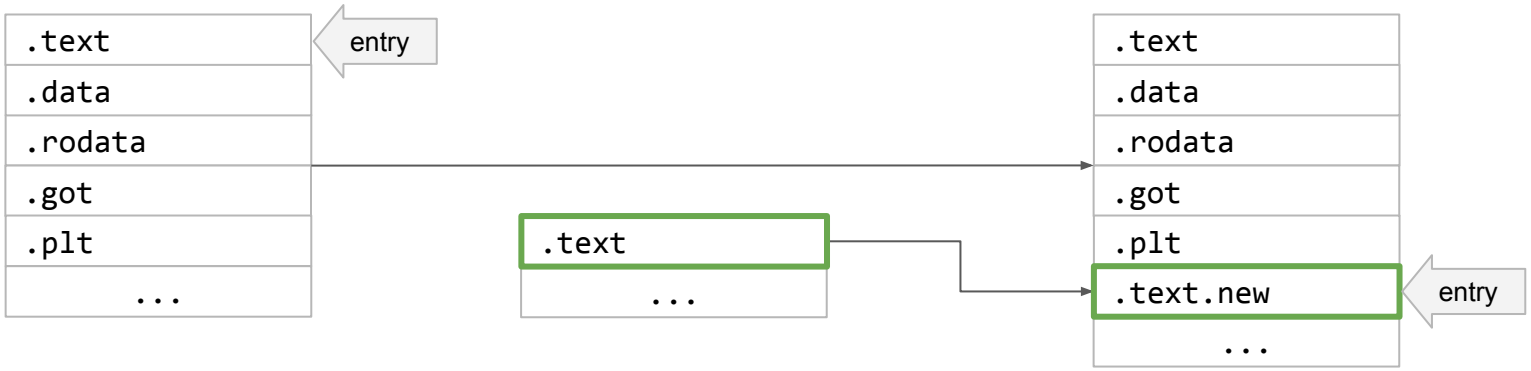

Cascading optimizations



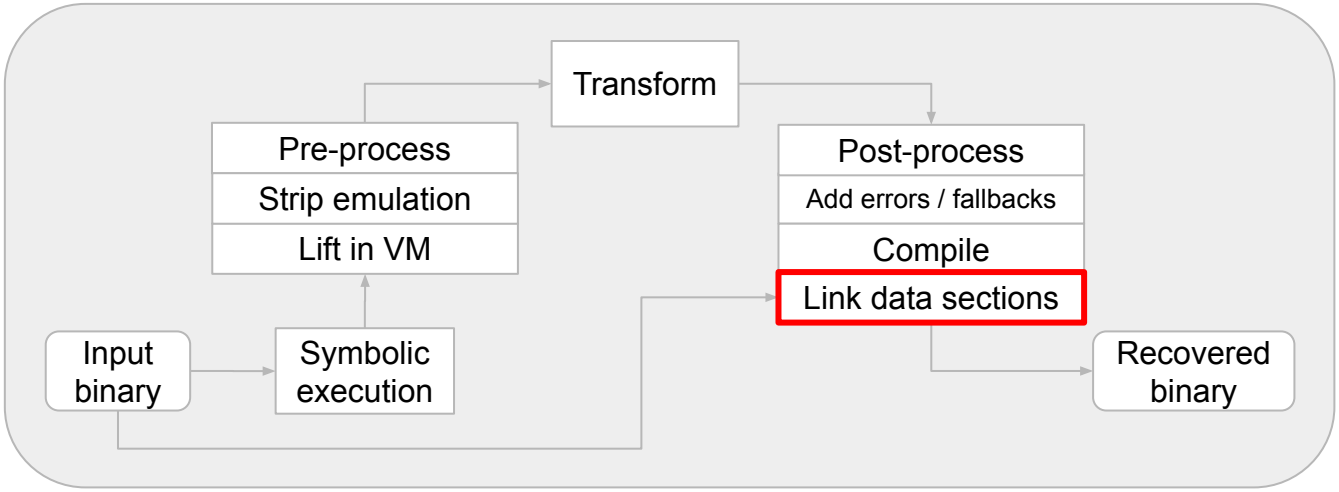
.text
...

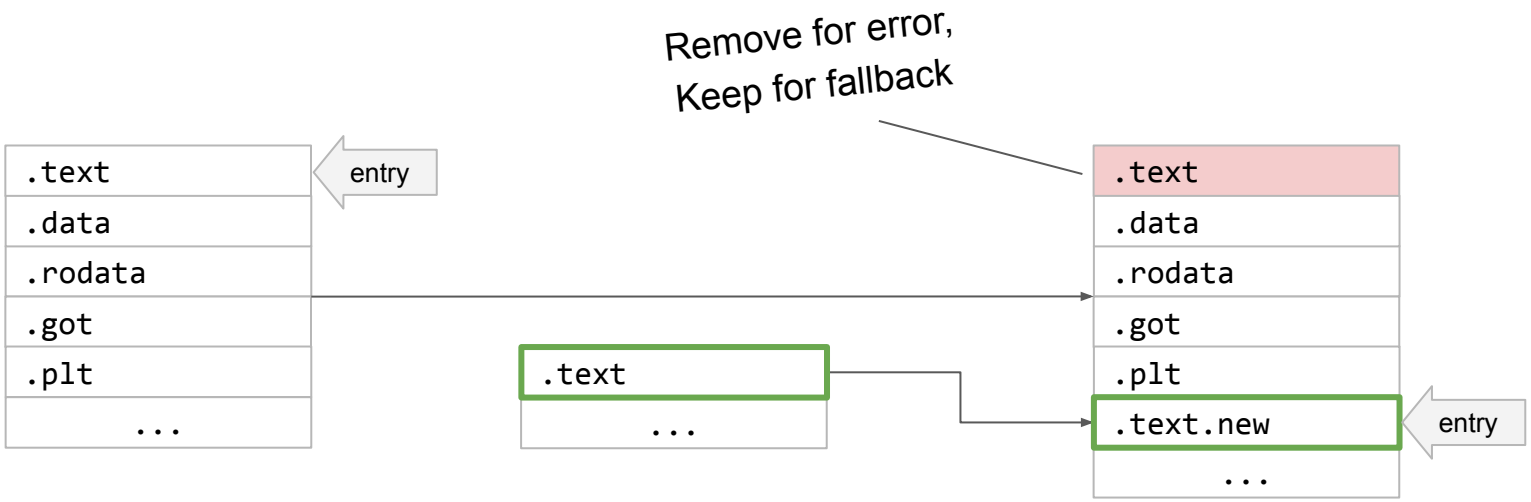
Recovered code



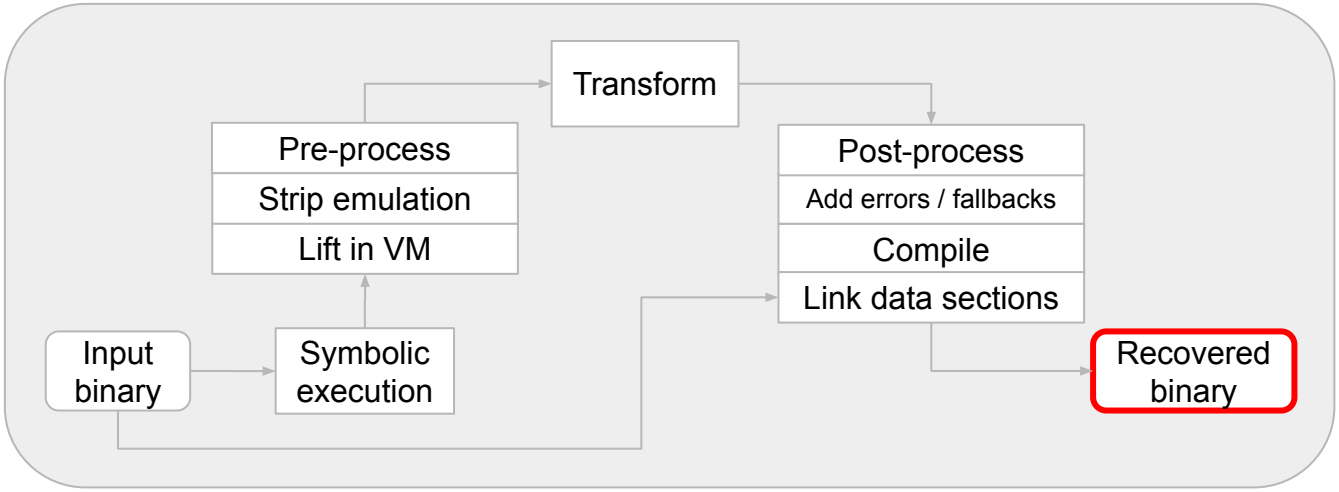


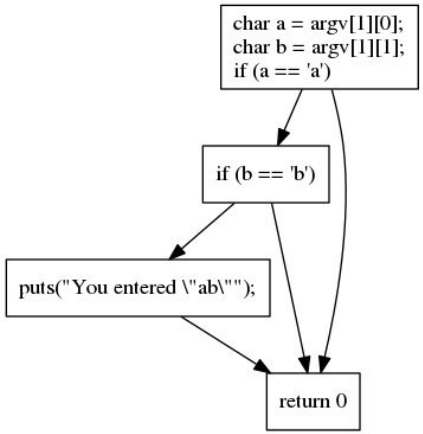
Old binary + Recovered code = Recovered binary



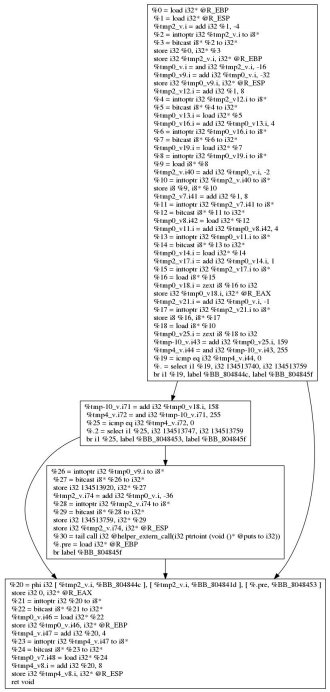


Old binary + Recovered code = Recovered binary

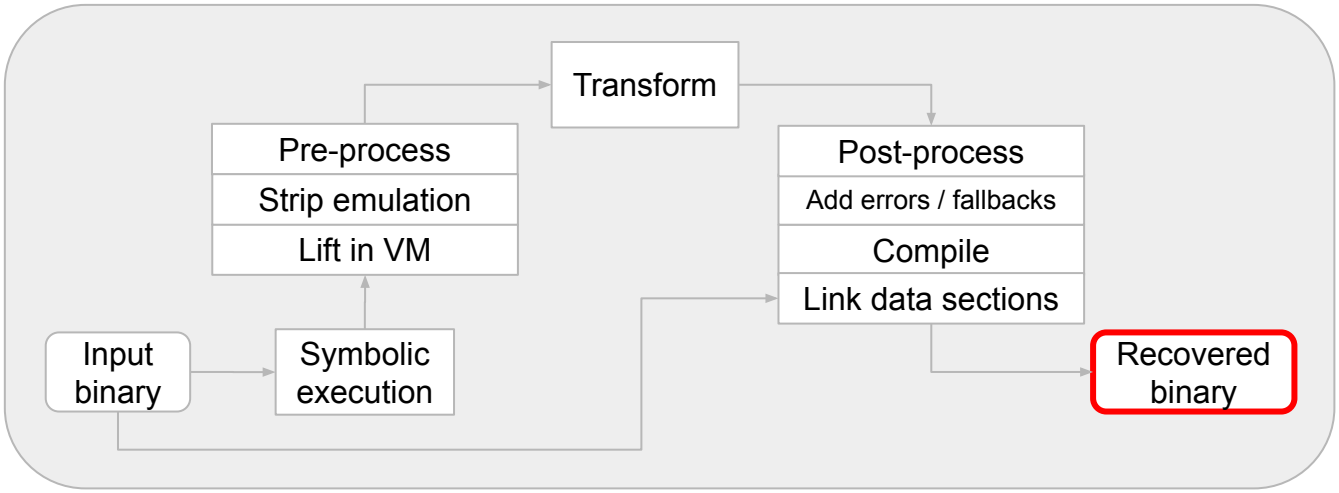


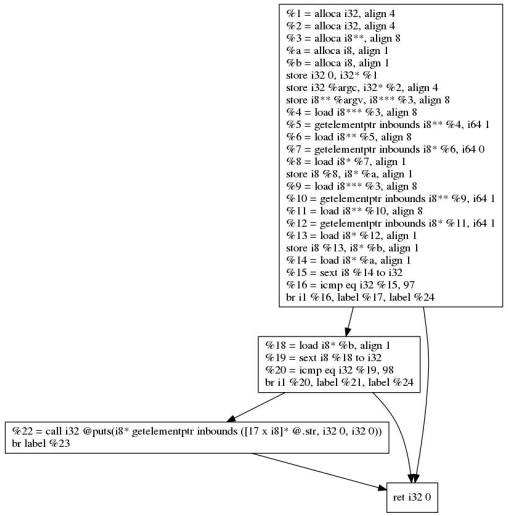


Original C

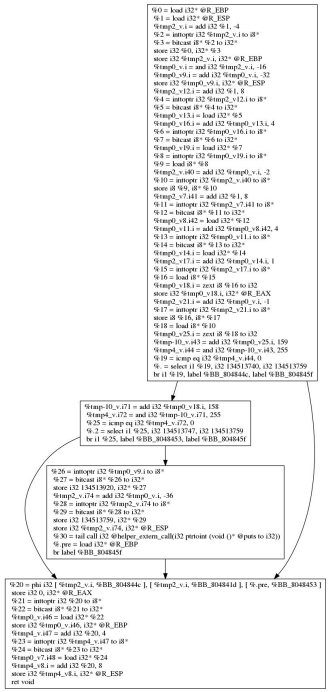


Recovered LLVM





Original LLVM



Recovered LLVM

Experiments

Experiments

- Correctness
- Attack Surface Reduction: ROP gadgets
- Performance

Experiment: correctness

Do our transformations preserve semantics?

- Yield errors for unknown code paths
- Check that recovered binary has same output as input binary

Experiment: correctness

Do our transformations preserve semantics?

- Yield errors for unknown code paths
- Check that recovered binary has same output as input binary

24 input binaries from SPEC-CPU2006 (x86)

- 15 succeeded, 9 failed (unexpected fallback / crash)

Experiment: ROP gadget reduction

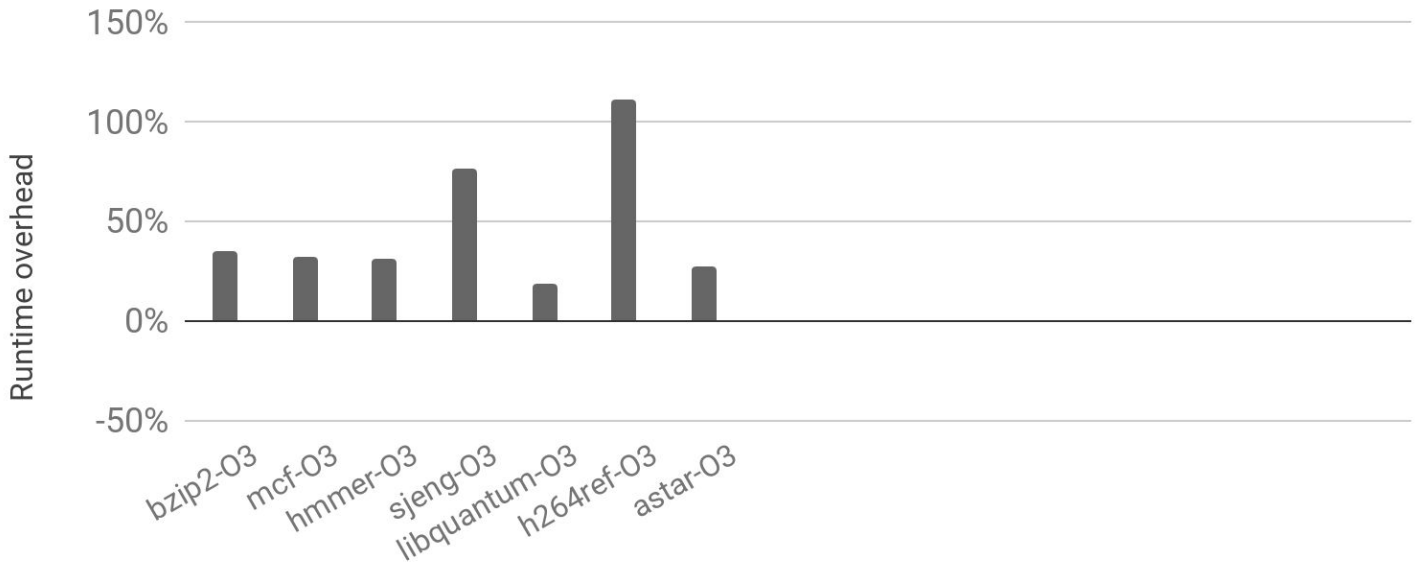
Is the attack surface actually smaller?

- 72% fewer instructions
- 48% fewer ROP gadgets

(both numbers are geomean)

Experiment: performance

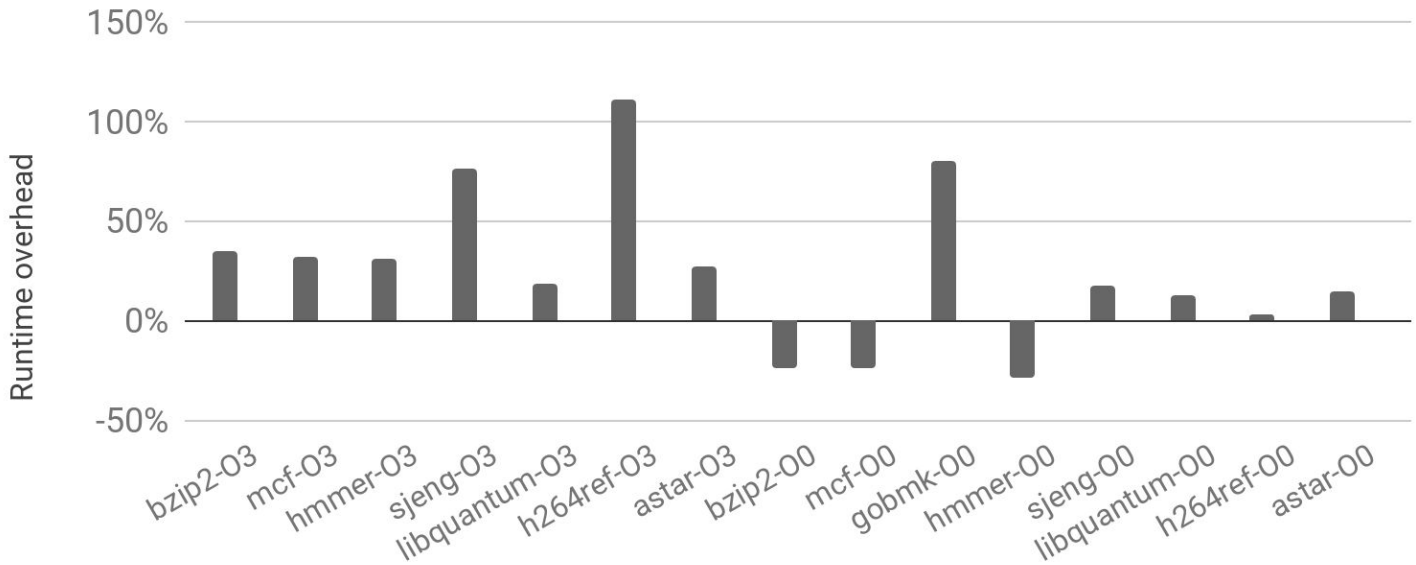
- `-O3` input binaries: expect similar performance
- `-O0` input binaries: expect speedup
- Disable fallback errors: maybe expect speedup



~44% overhead

Experiment: performance

- `-O3` input binaries: expect similar performance
- `-O0` input binaries: expect speedup
- Disable fallback errors: maybe expect speedup



~2% overhead

Experiment: performance

- `-O3` input binaries: expect similar performance
- `-O0` input binaries: expect speedup
- Disable fallback errors: maybe expect speedup

~5% performance gain

Wish list / future work

- Gadget-aware compiler backend
- Improve performance
  - Do aggressive profile-guided optimization
- Deobfuscation

Conclusion

- BinRec successfully transforms binaries at compiler IR level
- … and halves the ROP attack surface in the process

Also

- Binary lifting is hard