Encarsia:
Evaluating CPU Fuzzers via Automatic Bug Injection

Matej Bölcskei, Flavien Solt, Katharina Ceesay-Seitz, and Kaveh Razavi

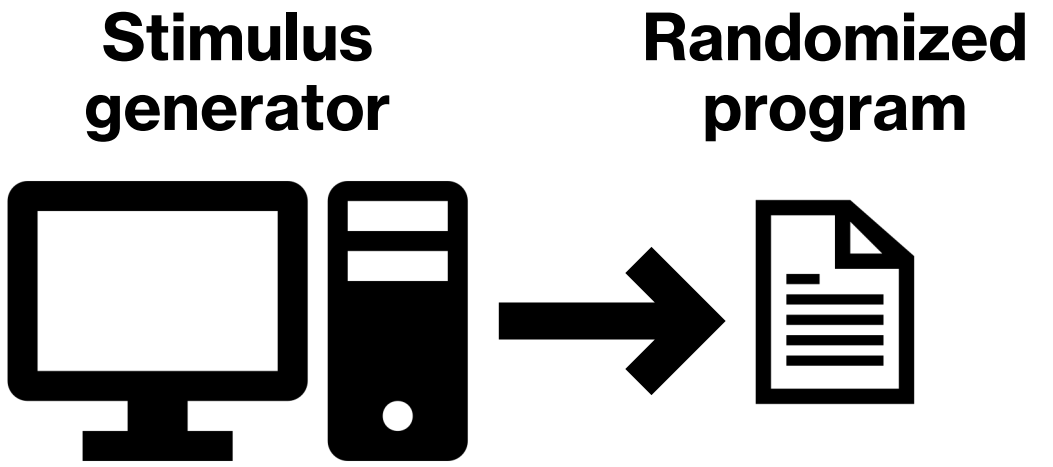
How do hardware fuzzers work?

How do hardware fuzzers work?

They are automated test benches

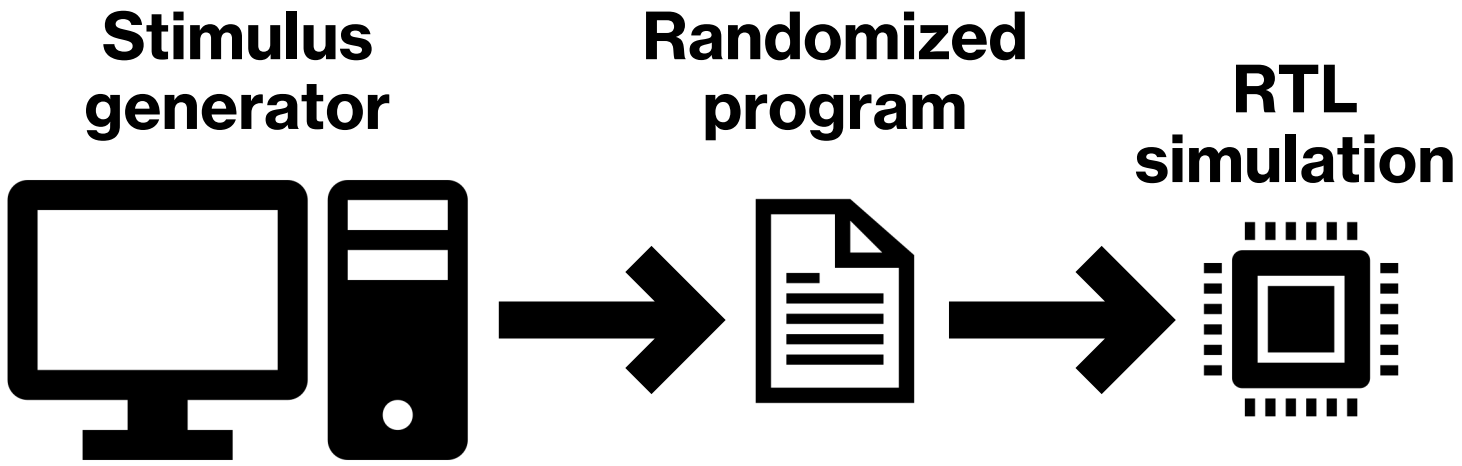
How do hardware fuzzers work?

They are automated test benches



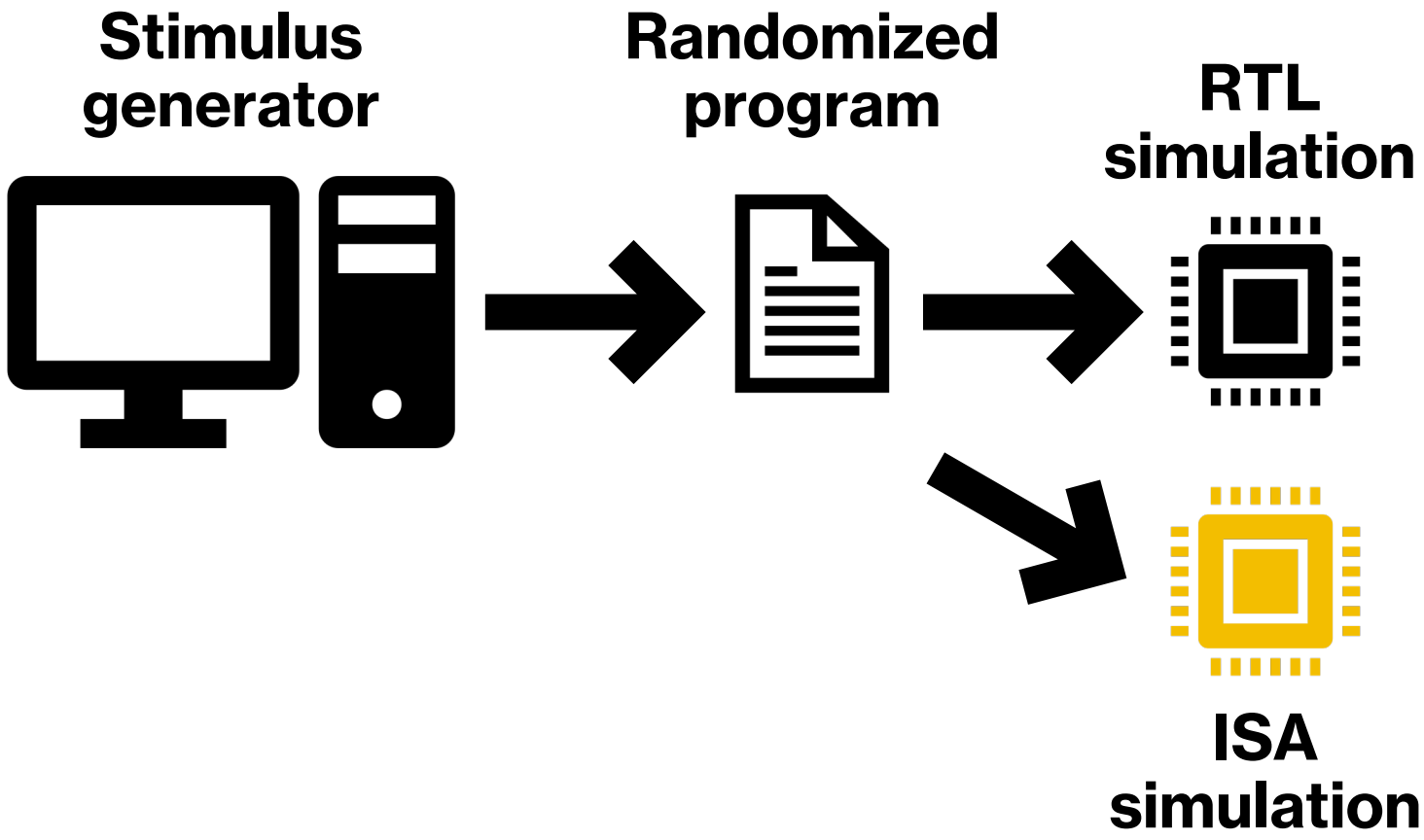
How do hardware fuzzers work?

They are automated test benches



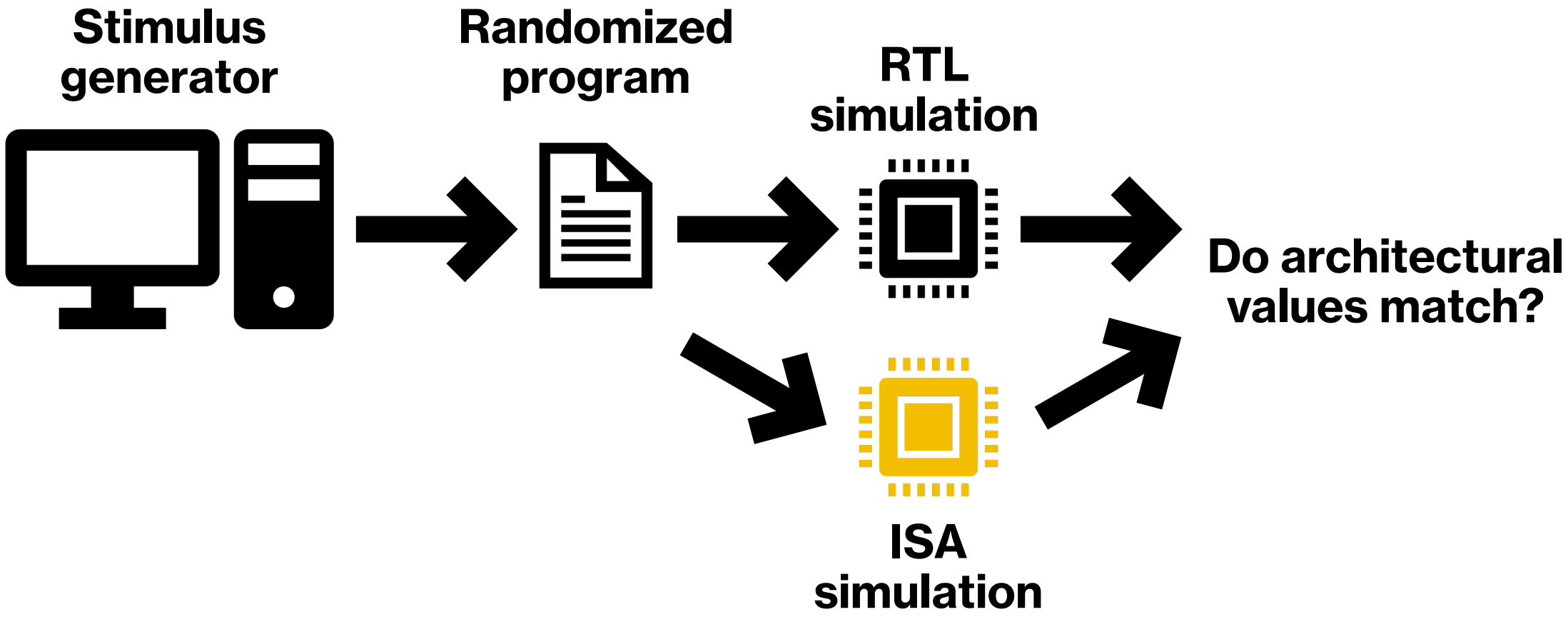
How do hardware fuzzers work?

They are automated test benches



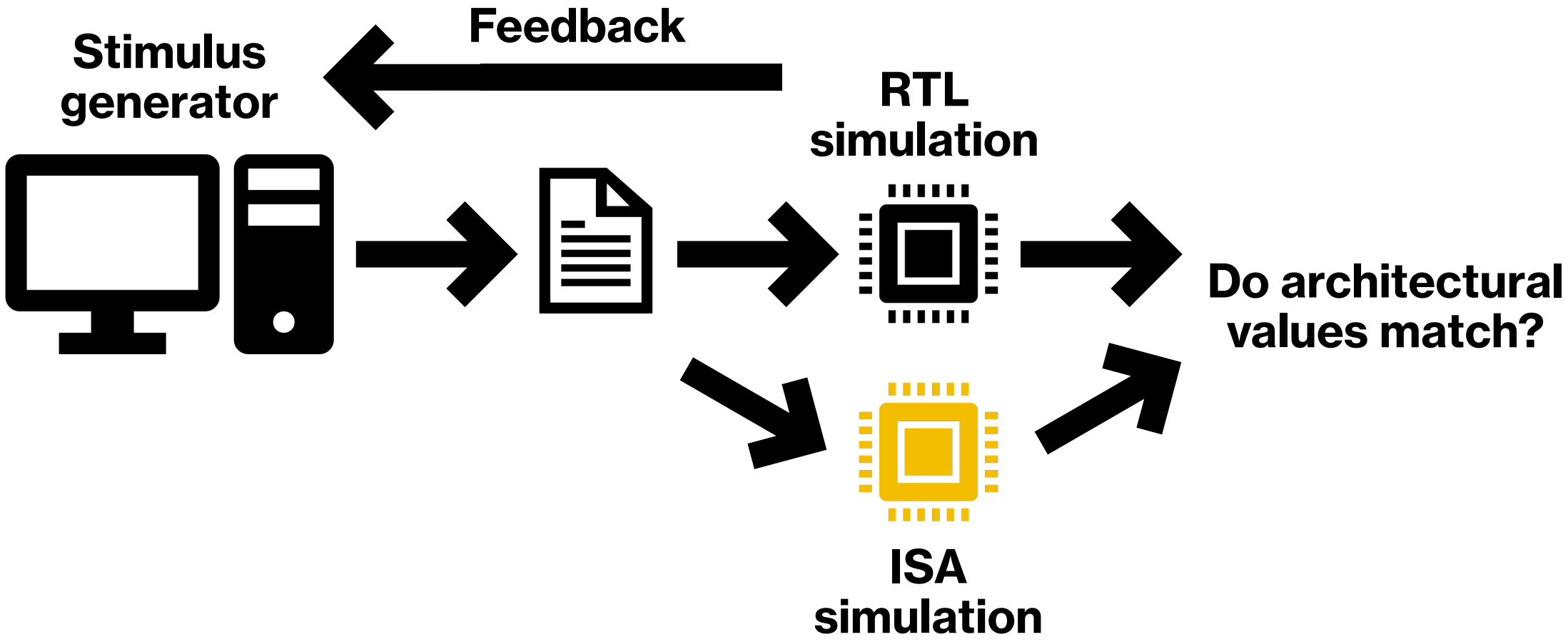
How do hardware fuzzers work?

They are automated test benches



How do hardware fuzzers work?

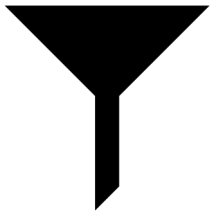
They are automated test benches



Fuzzers look great on paper

Fuzzers look great on paper

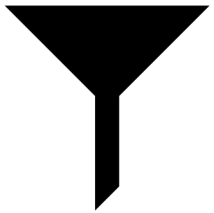
They achieve high coverage!



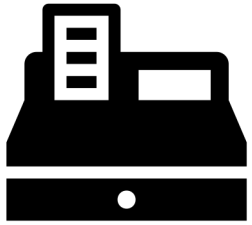
Multiplexer

Fuzzers look great on paper

They achieve high coverage!



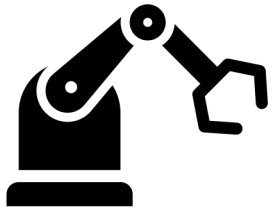
Multiplexer



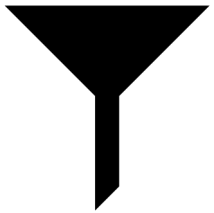
Register

Fuzzers look great on paper

They achieve high coverage!



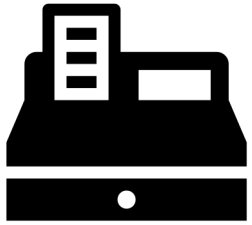
FSM



Multiplexer

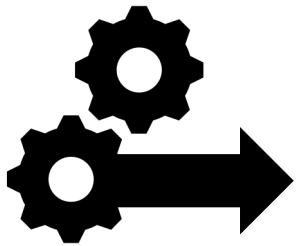
Edge

Branch



Register

Expression



Output

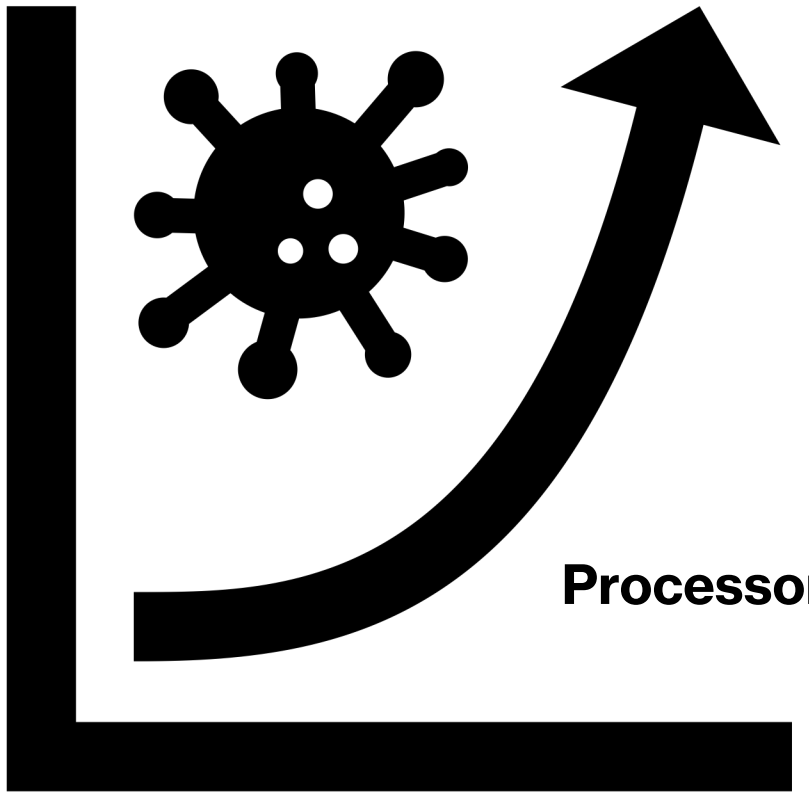
Statement

Fuzzers look great on paper

They find bugs!

Fuzzers look great on paper

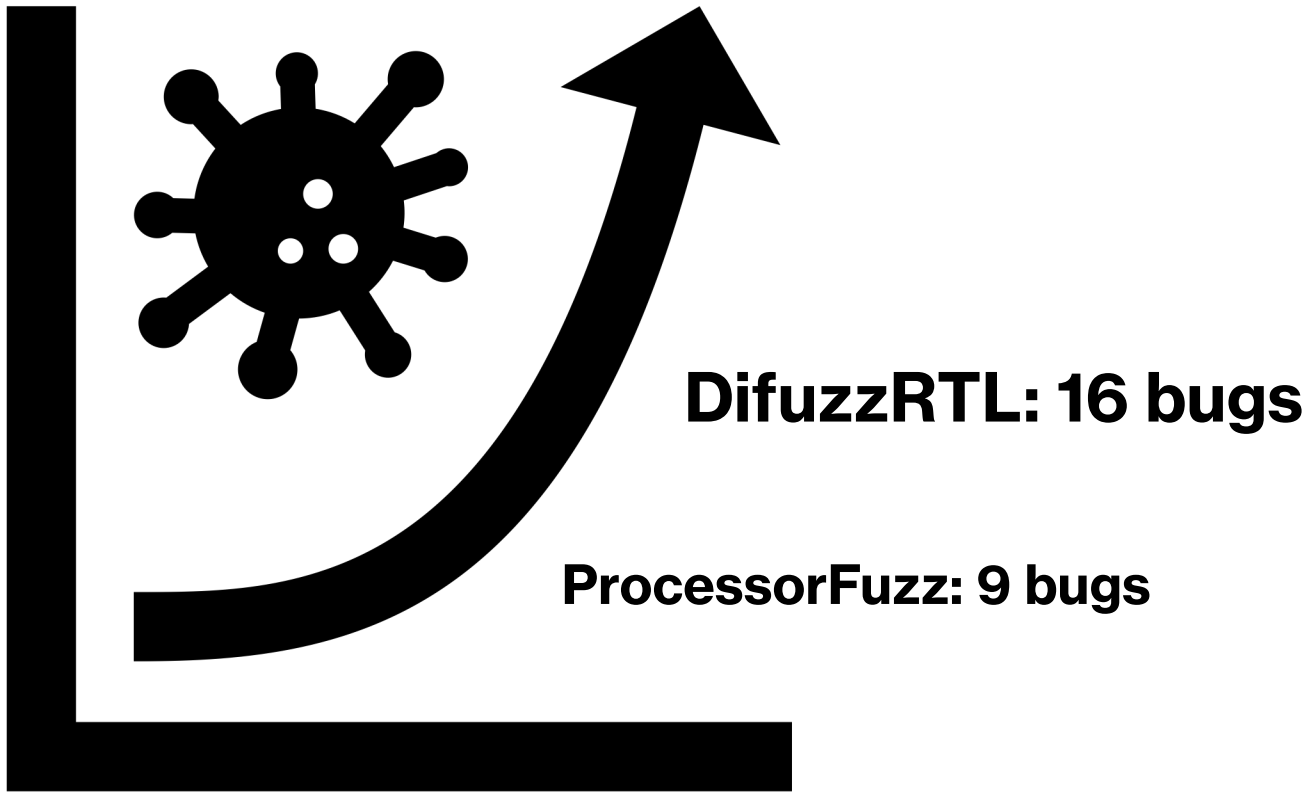
They find bugs!



ProcessorFuzz: 9 bugs

Fuzzers look great on paper

They find bugs!

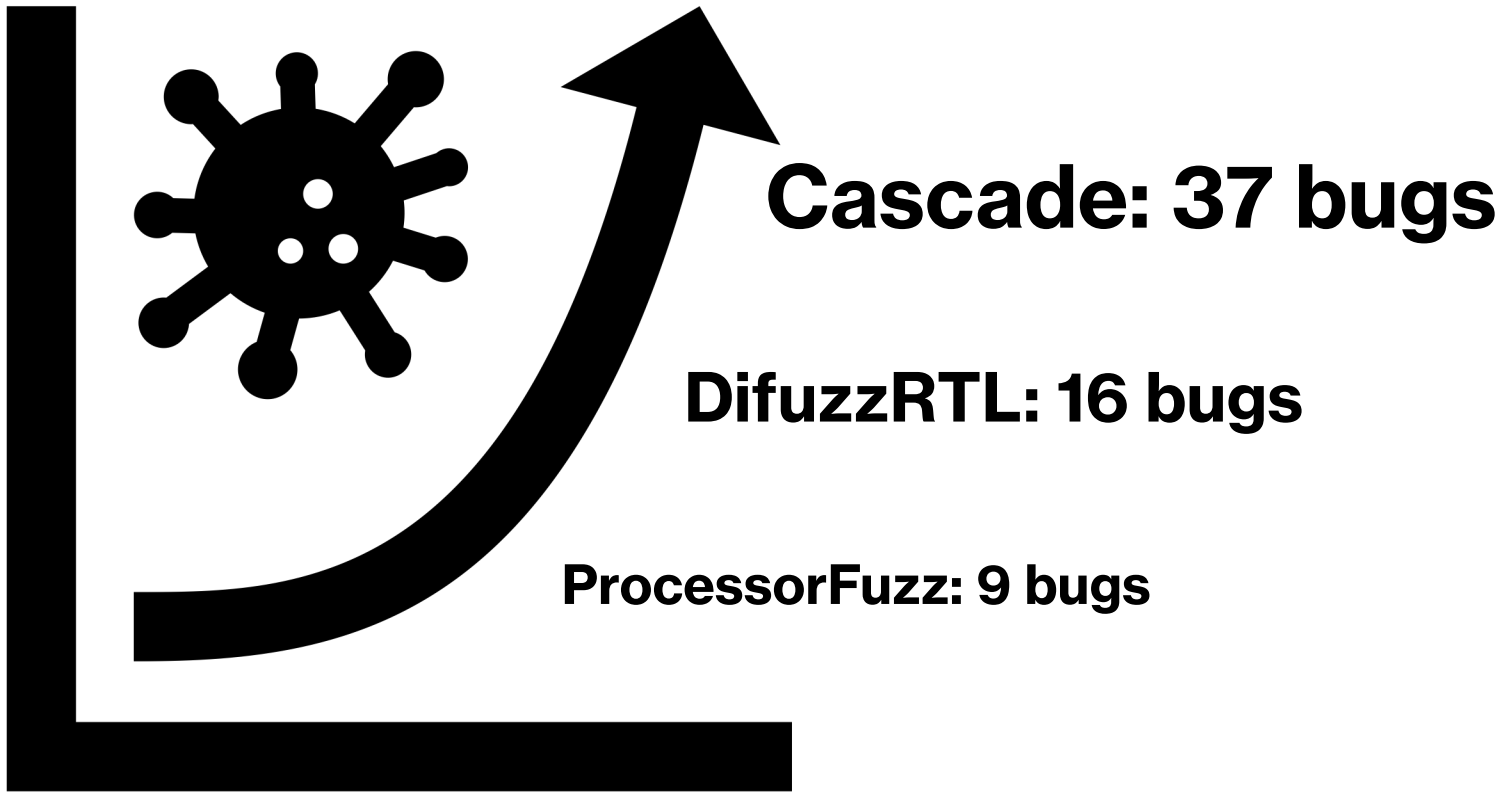


DifuzzRTL: 16 bugs

ProcessorFuzz: 9 bugs

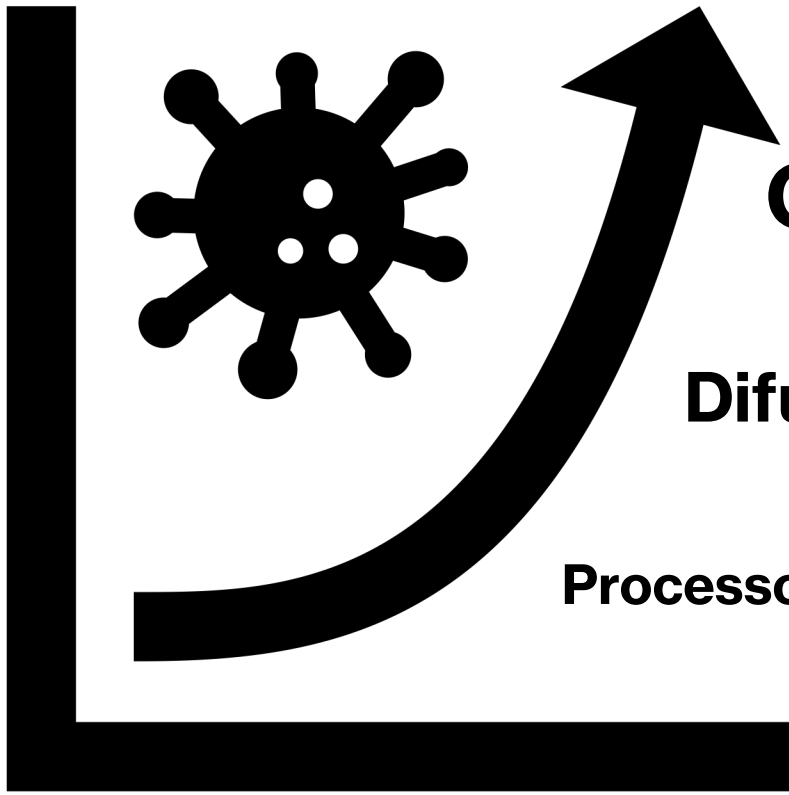
Fuzzers look great on paper

They find bugs!



Fuzzers look great on paper

They find bugs!



Everybody gets a bug!

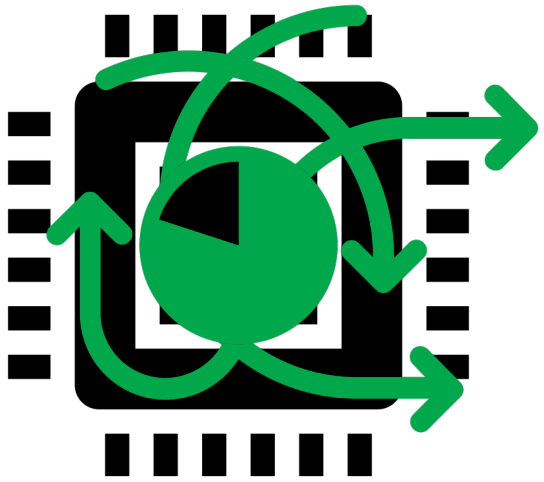
Cascade: 37 bugs

DifuzzRTL: 16 bugs

ProcessorFuzz: 9 bugs

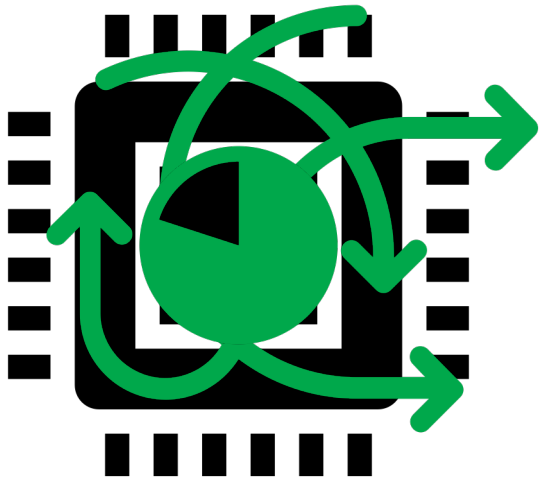
Creators use flashy figures to sell stories

Creators use flashy figures to sell stories



Design coverage

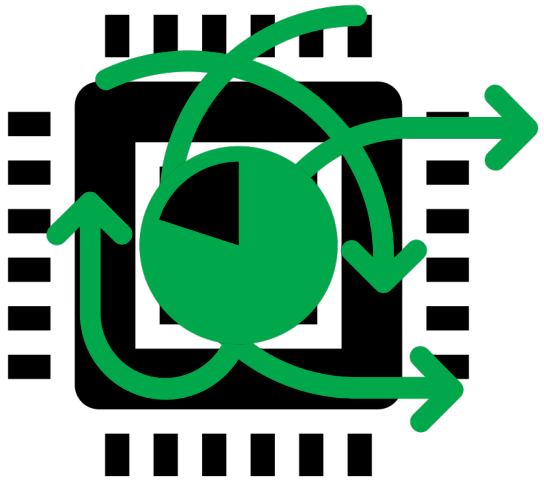
Creators use flashy figures to sell stories



Design coverage

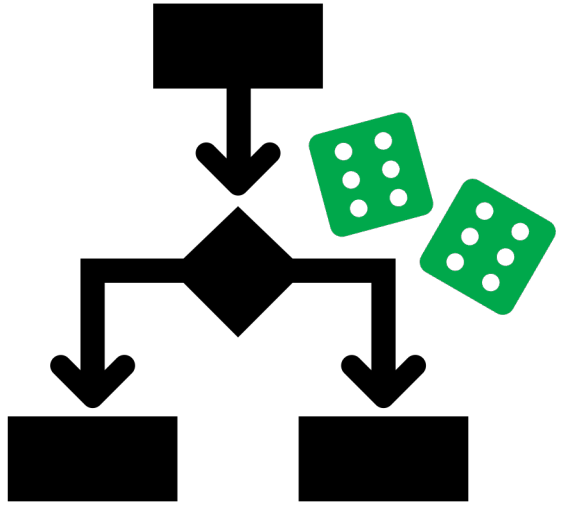
Bug detection

Creators use flashy figures to sell stories



Design coverage

Bug detection



Input generation

Fuzzer evaluation is flawed

Fuzzer evaluation is flawed

Coverage is not well suited for comparing fuzzers

L. Inozemtseva, and R. Holmes
“Coverage is not strongly correlated with test suite ...”
in ICSE 2014

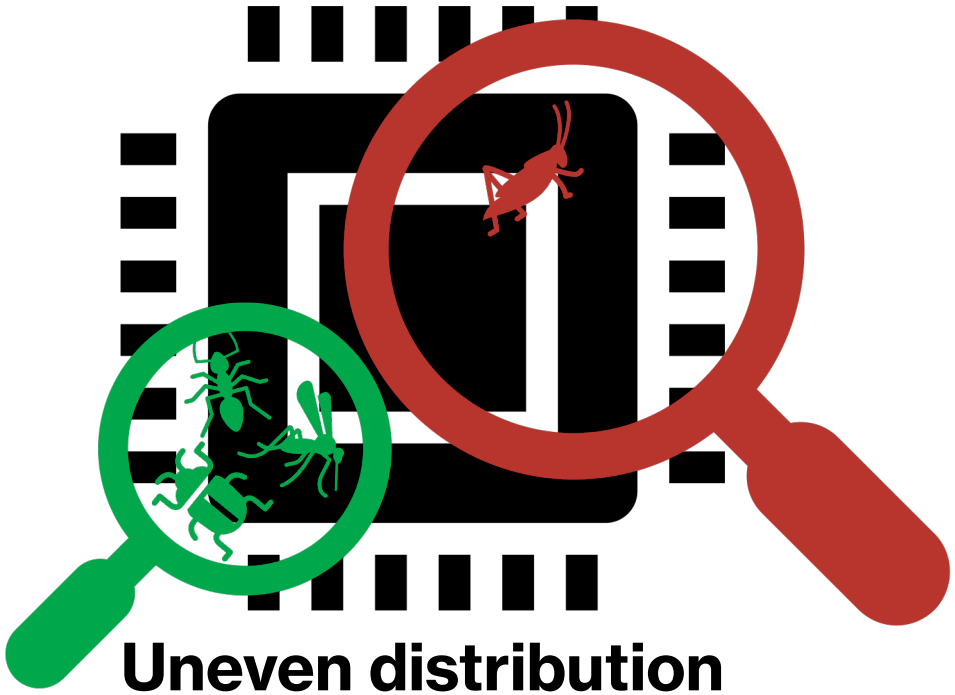
M. Böhme, L. Szekeres, and J. Metzman
“On the reliability of coverage-based fuzzer ...”
in ICSE 2022

Fuzzer evaluation is flawed

Coverage is not well suited for comparing fuzzers

L. Inozemtseva, and R. Holmes
“Coverage is not strongly correlated with test suite ...”
in ICSE 2014

M. Böhme, L. Szekeres, and J. Metzman
“On the reliability of coverage-based fuzzer ...”
in ICSE 2022



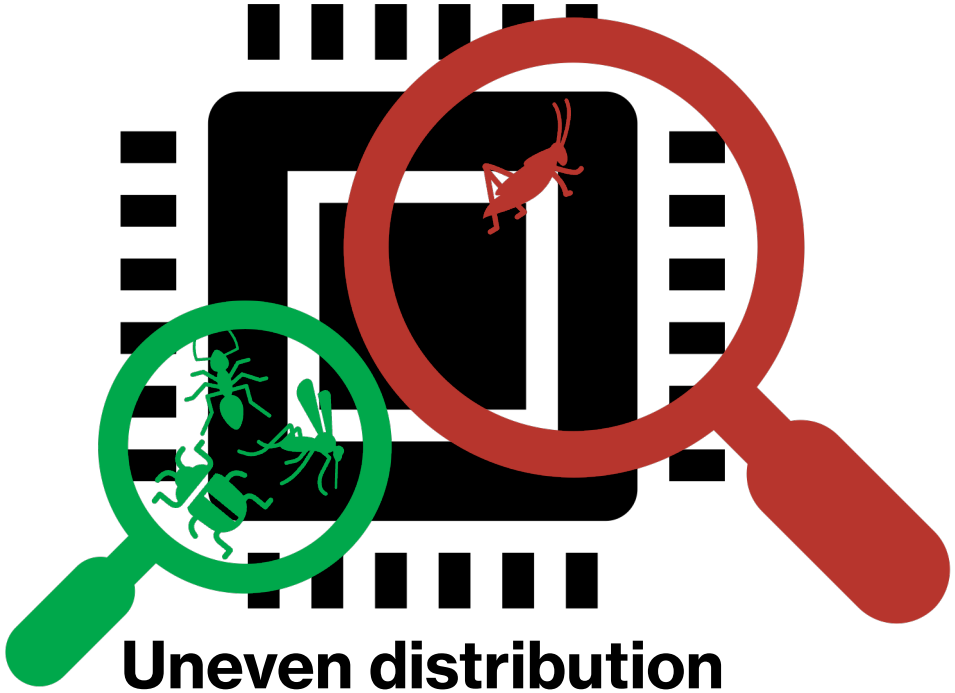
Uneven distribution

Fuzzer evaluation is flawed

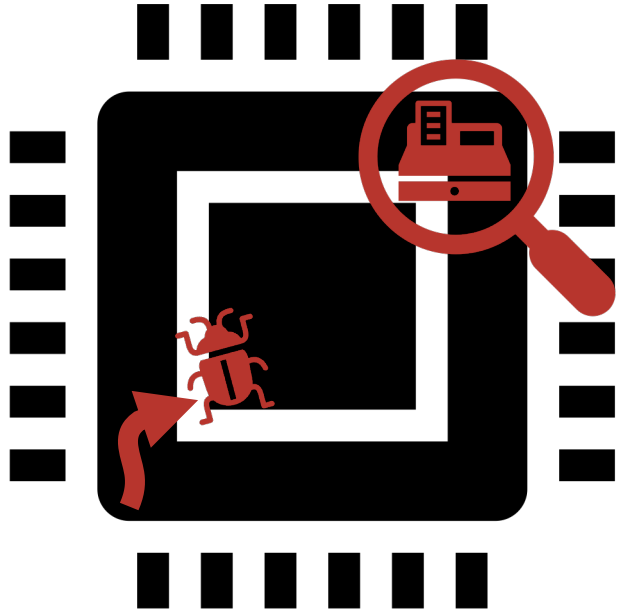
Coverage is not well suited for comparing fuzzers

L. Inozemtseva, and R. Holmes
“Coverage is not strongly correlated with test suite ...”
in ICSE 2014

M. Böhme, L. Szekeres, and J. Metzman
“On the reliability of coverage-based fuzzer ...”
in ICSE 2022



Uneven distribution



Ignores propagation

Fuzzer evaluation is flawed

Let’s examine bug discovery

Fuzzer evaluation is flawed

Let’s examine bug discovery

Bug	Device	Description
1	BlackParrot	Non-boxed single-precision floating point ...
2	BlackParrot	Read-after-Write dependencies on fcsr.fflags ...
3	BlackParrot	When mstatus.FS is not set and the fcsr is ...
4	BlackParrot	The 2 low-bits of sepc CSR are not ...
5	BlackParrot	No exception raised when writing certain ..
6	BlackParrot	Reading zero register, following specific ...
7	Dromajo	PMP checks are performed, and raise ...
8	Rocket & BOOM	Instruction page fault not raised when ...
9	BOOM	mstatus.FS is gratuitously set to dirty.

Fuzzer evaluation is flawed

Let's examine bug discovery

Bug	Device	Description
1	BlackParrot	Non-boxed single-precision floating point ...
2	BlackParrot	Read-after-Write dependencies on fcsr.fflags ...
3	BlackParrot	When mstatus.FS is not set and the fcsr is ...
4	BlackParrot	The 2 low-bits of sepc CSR are not ...
5	BlackParrot	No exception raised when writing certain ..
6	BlackParrot	Reading zero register, following specific ...
7	Dromajo	PMP checks are performed, and raise ...
8	Rocket & BOOM	Instruction page fault not raised when ...
9	BOOM	mstatus.FS is gratuitously set to dirty.

Never Fuzzed

Flawed evaluation fractures the field

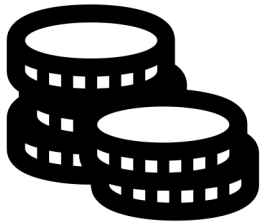
Flawed evaluation fractures the field

What do researchers think of coverage?

Flawed evaluation fractures the field

What do researchers think of coverage?

J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee
“Difuzzrtl: Differential fuzz testing to find cpu bugs”
in S&P 2021



Cheap

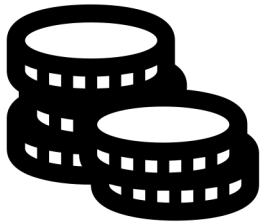
Effective

Flawed evaluation fractures the field

What do researchers think of coverage?

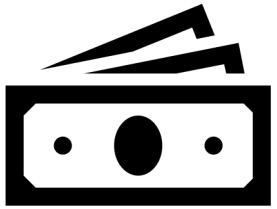
J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee
“Difuzzrtl: Differential fuzz testing to find cpu bugs”
in S&P 2021

F. Solt, K. Ceesay-Seitz, and K. Razavi
“Cascade: CPU Fuzzing via Intricate Program Generation”
in USENIX Security 2024



Cheap

Effective



Expensive

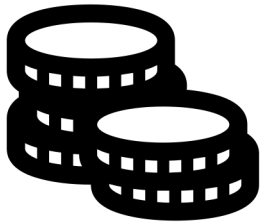
Ineffective

Flawed evaluation fractures the field

What do researchers think of coverage?

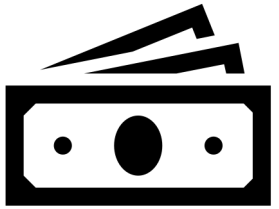
J. Hur, S. Song, D. Kwon, E. Baek, J. Kim, and B. Lee
“Difuzzrtl: Differential fuzz testing to find cpu bugs”
in S&P 2021

F. Solt, K. Ceesay-Seitz, and K. Razavi
“Cascade: CPU Fuzzing via Intricate Program Generation”
in USENIX Security 2024



Cheap

Effective

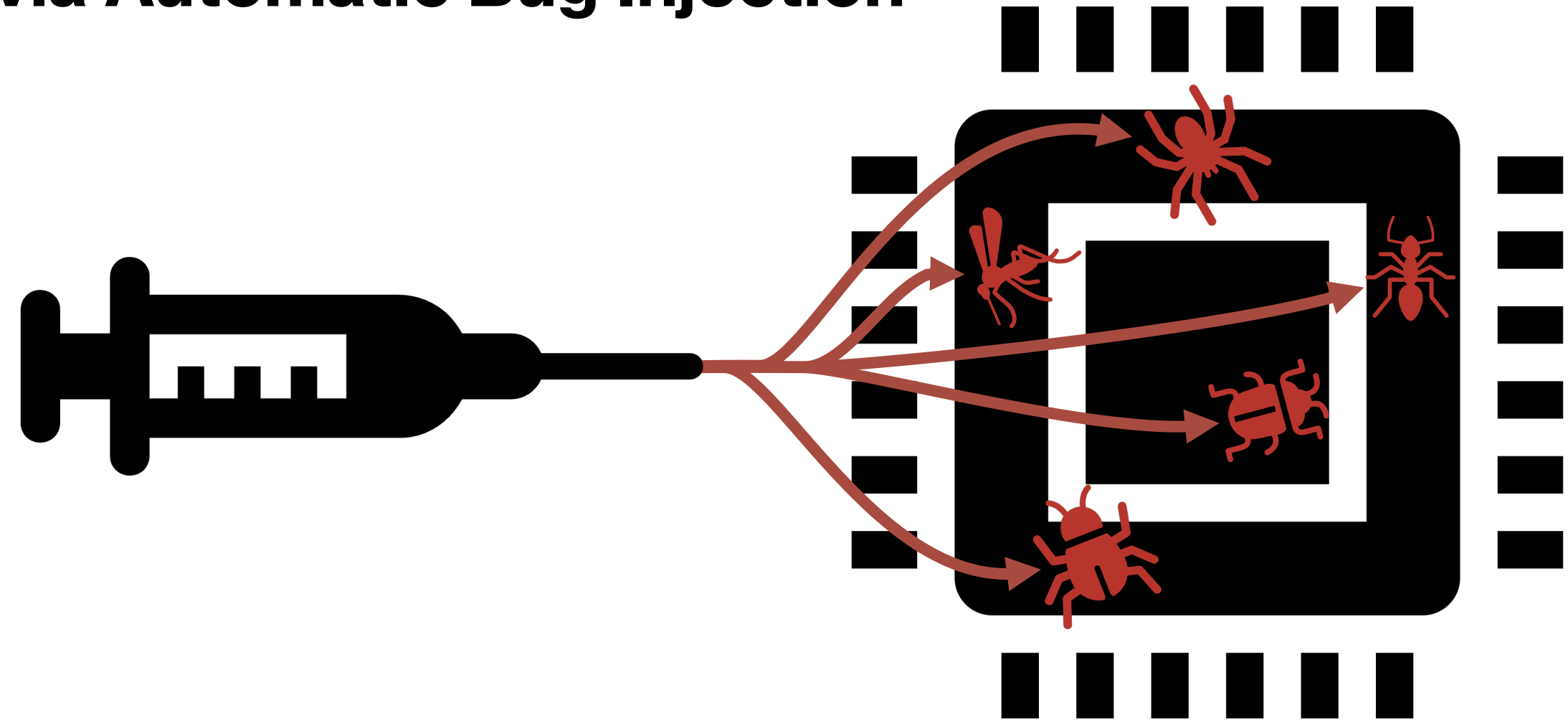


Expensive

Ineffective

Contradiction!

Encarsia: Evaluating CPU Fuzzers via Automatic Bug Injection



How do bugs arise in the real world?

How do bugs arise in the real world?

We survey 1672 pull requests

#399 Prevents illegal CSR writes

Accessing a debug register outside of debug mode sets illegal_csr, but the CSR write still happened. This change ensures that if an illegal CSR instruction is signaled, the CSR write is blocked.

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--  csr_wreq & ~illegal_csr_priv & instr_new_id_i;
++  csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
```

How do bugs arise in the real world?

We survey 1672 pull requests

#399 Prevents illegal CSR writes

```
Accessing a debug register outside of debug mode sets
illegal_csr, but the CSR write still happened. This
change ensures that if an illegal CSR instruction is
signaled, the CSR write is blocked.
```

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--  csr_wreq & ~illegal_csr_priv & instr_new_id_i;
++  csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
```

Source	Conditionals	Mix-ups	Other
Ibex	17	14	0
CVA6	22	11	0
Rocket	8	41	0
BOOM	13	33	0
DAC19	2	2	0
DAC21	4	10	0
Total	66	111	0

All 177 bugs fall into 2 categories

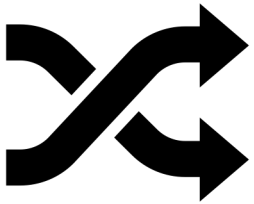
How do bugs arise in the real world?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--   csr_wreq & ~illegal_csr_priv & instr_new_id_i;
++   csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
```

Definition



Confusing similar signals

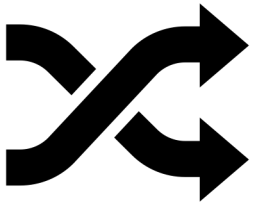
How do bugs arise in the real world?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--   csr_wreq & ~illegal_csr_priv & instr_new_id_i;
++   csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
```

Definition

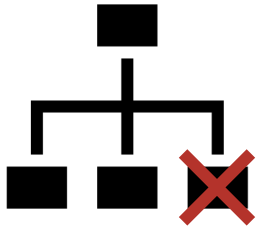


Confusing similar signals

Broken Conditionals

```
illegal_csr   = 1'b0;
case (csr_addr_i)
    CSR_MSTATUS: csr_rdata_int = mstatus_q
    CSR_DCSR: begin
        csr_rdata_int = dcsr_q;
++      illegal_csr = ~debug_mode_i;
    end
    default: illegal_csr = 1'b1;
endcase
```

Definition



Mishandling corner cases

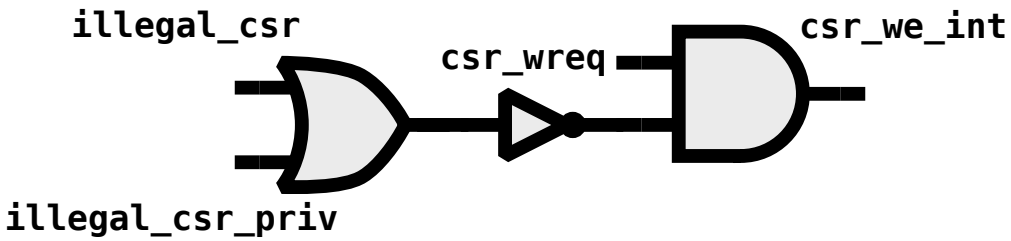
How do we inject bugs?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--  csr_wreq & ~illegal_csr_priv & instr_new_id_i;
++  csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
```

Intermediate representation



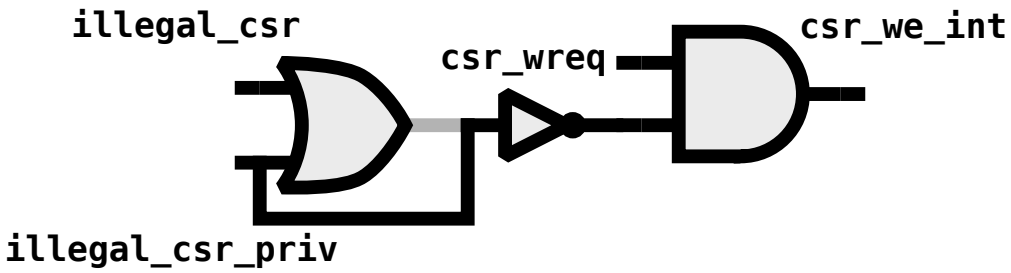
How do we inject bugs?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--   csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
++   csr_wreq & ~illegal_csr_priv & instr_new_id_i;
```

Intermediate representation



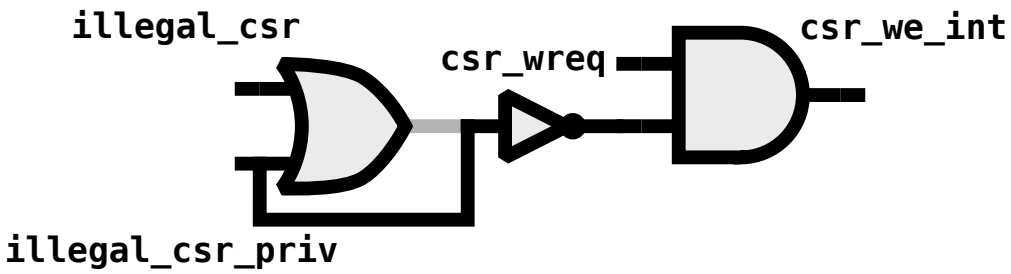
How do we inject bugs?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--  csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
++  csr_wreq & ~illegal_csr_priv & instr_new_id_i;
```

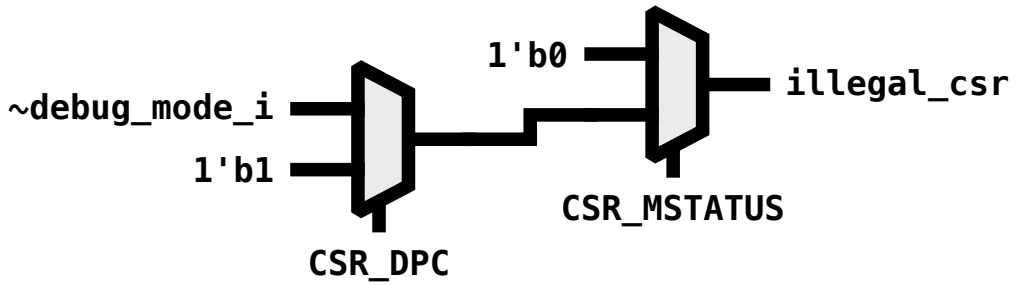
Intermediate representation



Broken Conditionals

```
illegal_csr   = 1'b0;
case (csr_addr_i)
    CSR_MSTATUS: csr_rdata_int = mstatus_q
    CSR_DCSR: begin
        csr_rdata_int = dcsr_q;
++      illegal_csr = ~debug_mode_i;
    end
    default: illegal_csr = 1'b1;
endcase
```

Intermediate representation



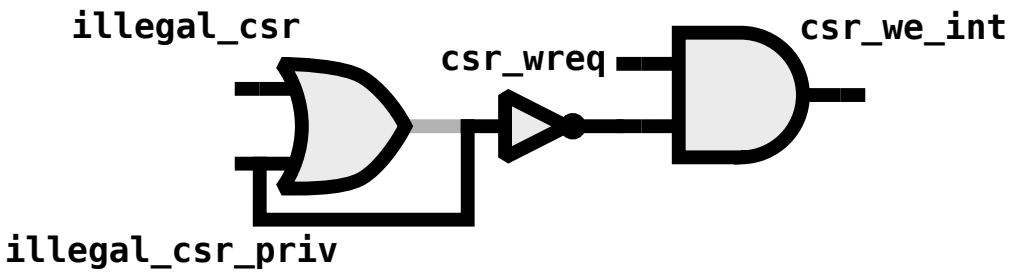
How do we inject bugs?

Signal Mix-ups

```
assign illegal_csr_insn_o =
    illegal_csr | illegal_csr_priv;

assign csr_we_int =
--  csr_wreq & ~illegal_csr_insn_o & instr_new_id_i;
++  csr_wreq & ~illegal_csr_priv & instr_new_id_i;
```

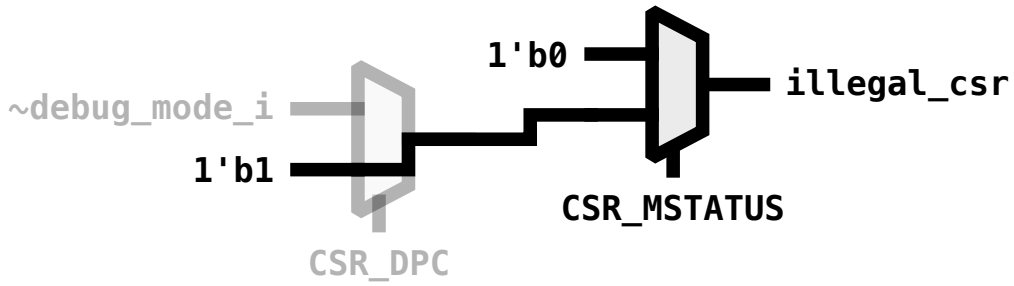
Intermediate representation



Broken Conditionals

```
illegal_csr   = 1'b0;
case (csr_addr_i)
    CSR_MSTATUS: csr_rdata_int = mstatus_q
    CSR_DCSR: begin
        csr_rdata_int = dcsr_q;
--      illegal_csr = ~debug_mode_i;
    end
    default: illegal_csr = 1'b1;
endcase
```

Intermediate representation

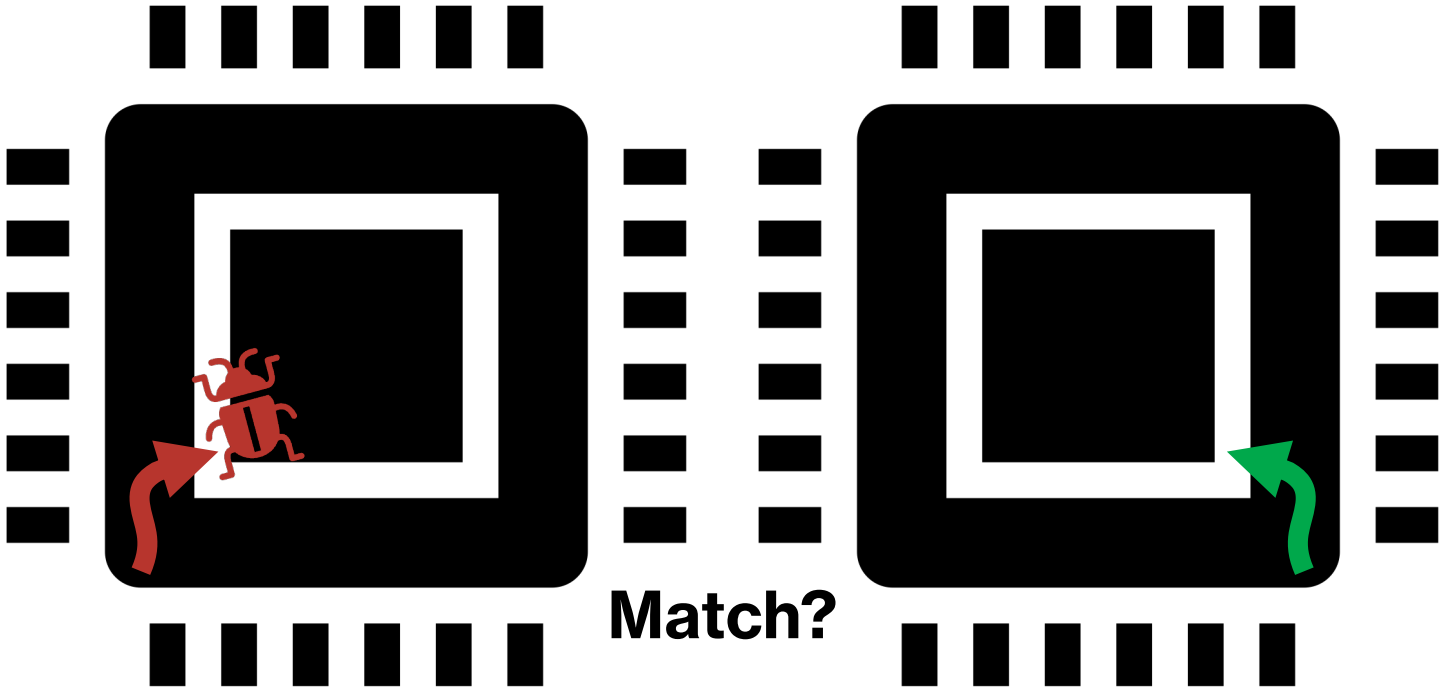


What about ineffective transformations?

We formally verify architectural observability

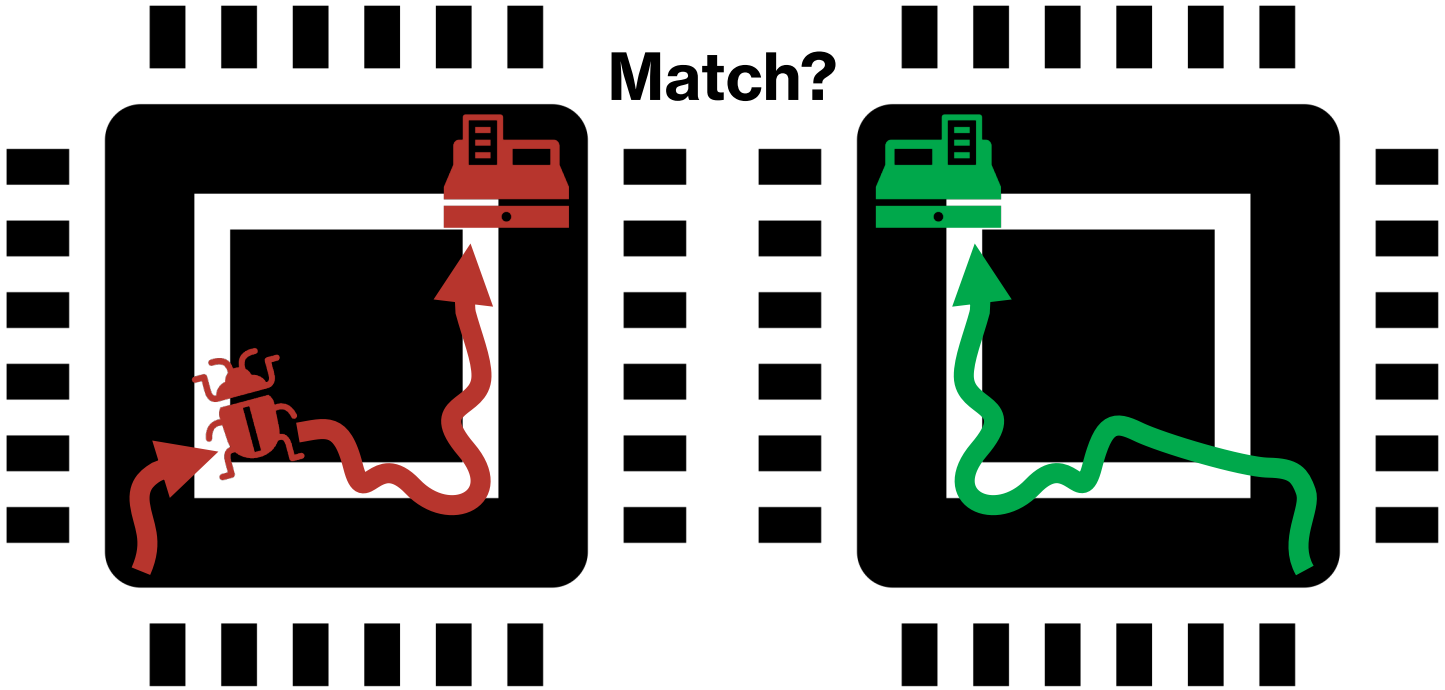
What about ineffective transformations?

We formally verify architectural observability



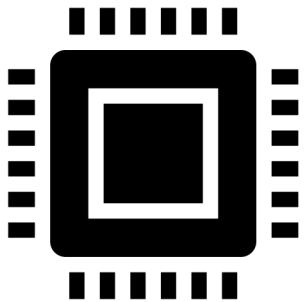
What about ineffective transformations?

We formally verify architectural observability

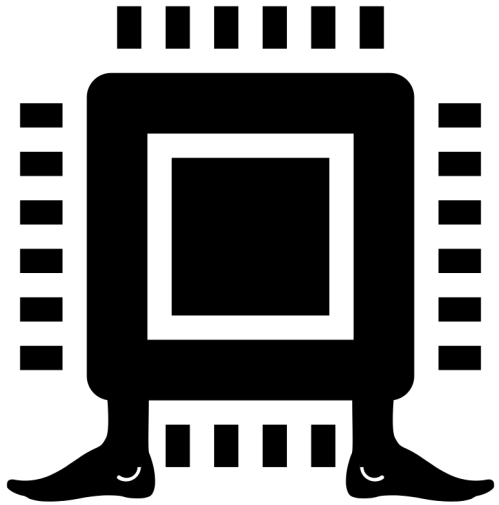


Which fuzzing techniques actually work?

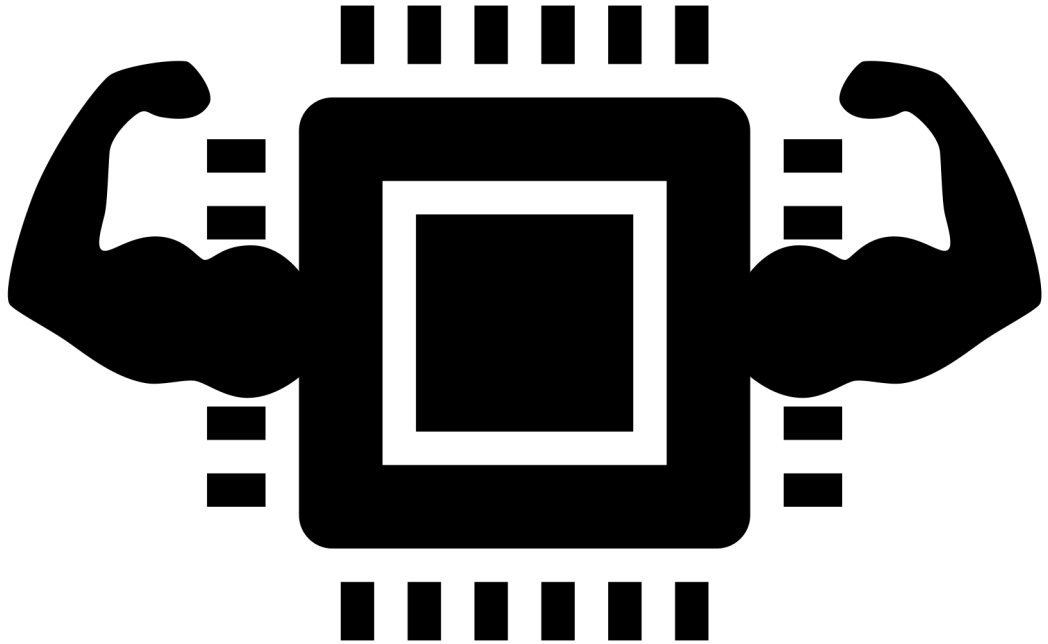
We generate EnCorpus



Ibex



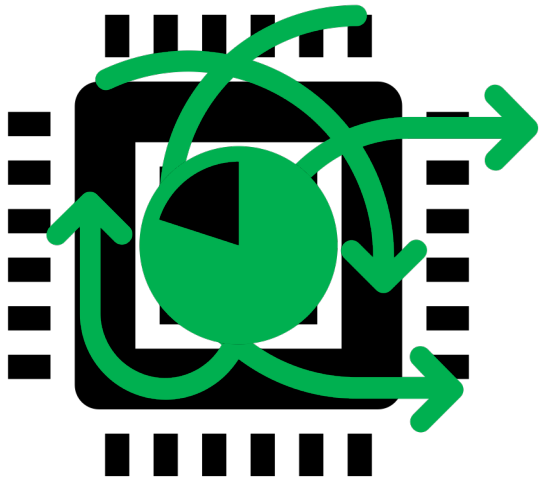
Rocket



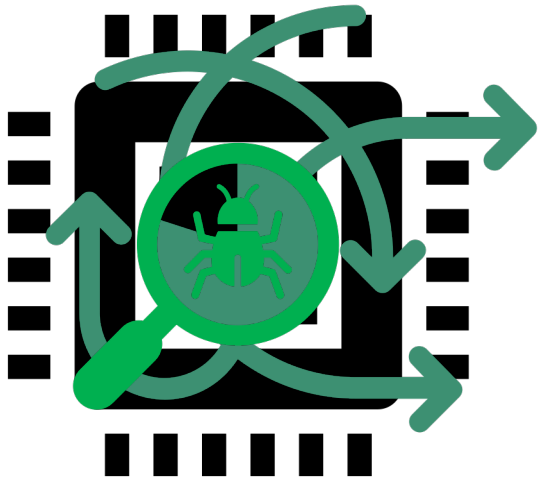
BOOM

Which fuzzing techniques actually work?

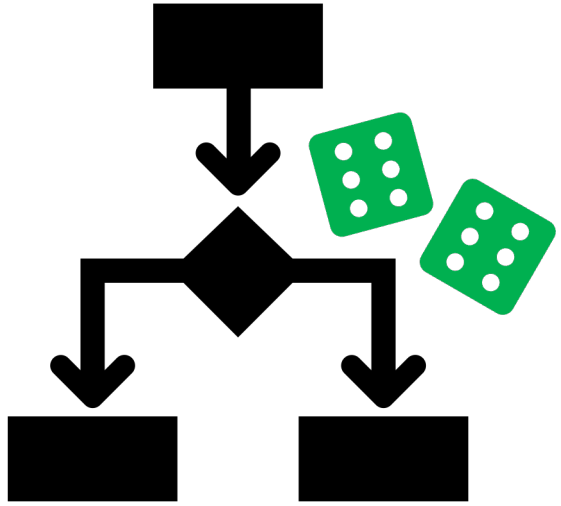
We must test each technique in isolation



DifuzzRTL

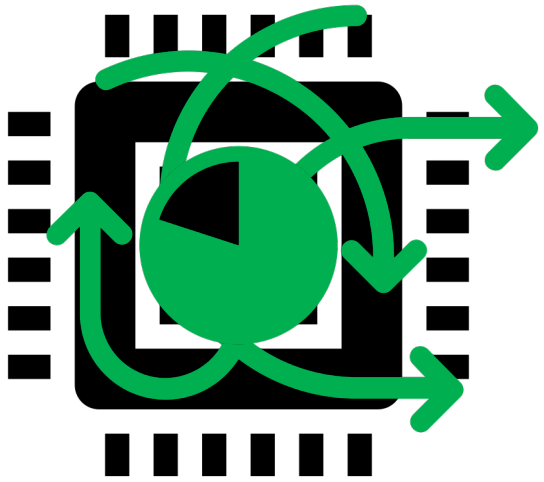


ProcessorFuzz

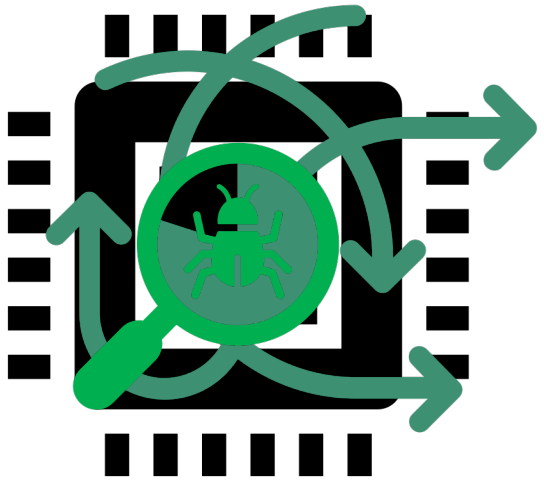


Cascade

Is instruction-granular detection worth it?

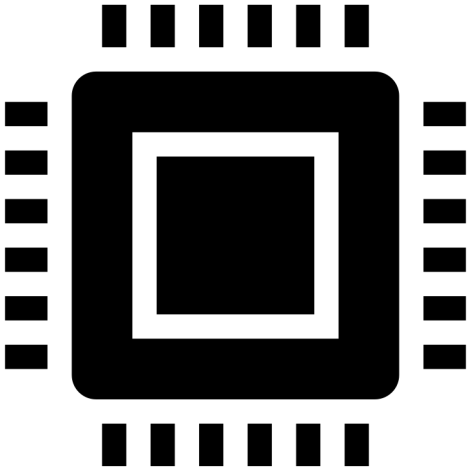


DifuzzRTL



ProcessorFuzz

Is instruction-granular detection worth it?



DifuzzRTL

ProcessorFuzz

Is instruction-granular detection worth it?

Rocket

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
PFuzz	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
PFuzz	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓

Is instruction-granular detection worth it?

Rocket

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
PFuzz	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
PFuzz	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

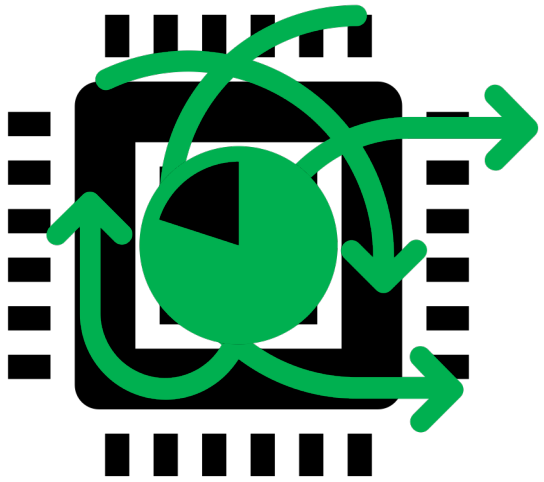
Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓

Broken Conditionals

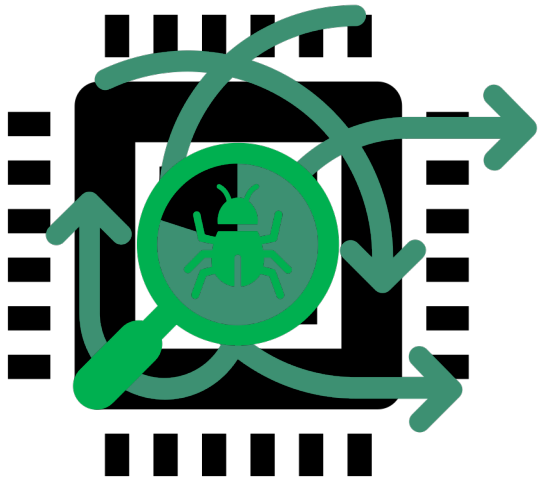
Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓

Insight 1:
Finer detection granularity does not yield better results.

Are hardware coverage metrics effective?



DifuzzRTL



ProcessorFuzz

Are hardware coverage metrics effective?

Rocket

Signal Mix-ups

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
PFuzz	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
PFuzz	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓

Broken Conditionals

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓

Are hardware coverage metrics effective?

Rocket

Signal Mix-ups

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
PFuzz	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
PFuzz	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

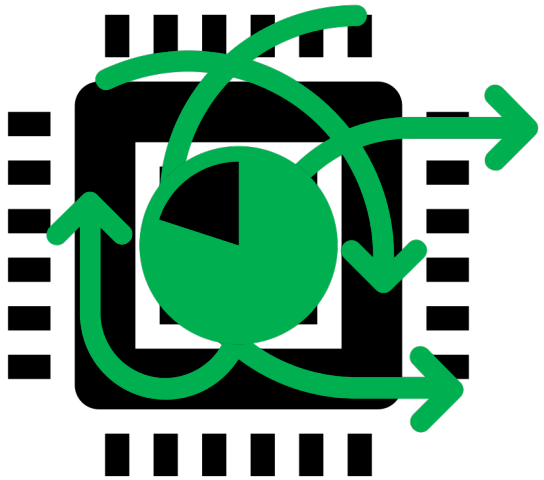
Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓

Broken Conditionals

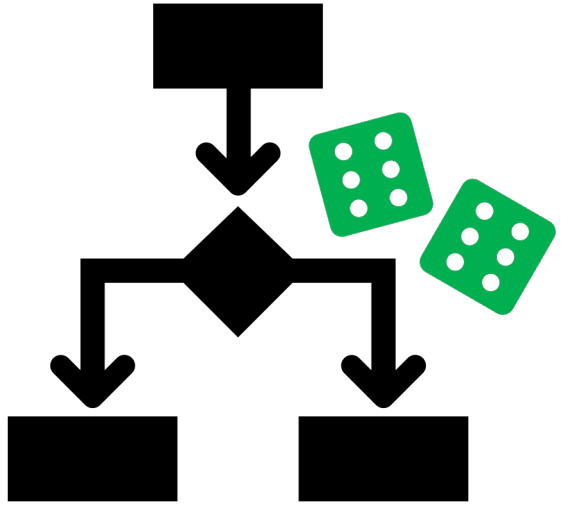
Cover.	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
None	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
PFuzz	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓

Insight 2:
Coverage metrics do not effectively guide fuzzers.

Does program generation play a role?



DifuzzRTL



Cascade

Does program generation play a role?

Rocket

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
Cascade	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
Cascade	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
Cascade	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
Cascade	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗

Does program generation play a role?

Rocket

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
Cascade	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✓	✗	✗	✓
Cascade	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓

BOOM

Signal Mix-ups

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✗	✗	✓
Cascade	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓

Broken Conditionals

Fuzzer	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DifRTL	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✓
Cascade	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗

Insight 3:
Program generation is the primary driver of finding bugs.

Encarsia helps detect misconfigurations

DifuzzRTL

Bug	Detected	Module
5	✓	UOPCodeFDivDec ...
6	✗	DivUnit
7	✗	MulDiv
8	✓	IntToFP
9	✗	PipelinedMulUnit
10	✓	ALU
11	✓	RegisterReadDecode

Encarsia helps detect misconfigurations

DifuzzRTL

Bug	Detected	Module
5	✓	UOPCodeFDivDec ...
6	✗	DivUnit
7	✗	MulDiv
8	✓	IntToFP
9	✗	PipelinedMulUnit
10	✓	ALU
11	✓	RegisterReadDecode

Forgets the M extension

```
def __init__(self, isa='RV64G'):
    isas = []
    if 'I' in isa:
        isas += [ 'rv32i' ]
    if 'M' in isa:
        isas += [ 'rv32m' ]
    if 'A' in isa:
        isas += [ 'rv32a' ]
    if 'F' in isa:
        isas += [ 'rv32f' ]
    if 'G' in isa:
--      isas = ['rv32i', 'rv32m', 'rv32a', 'rv32f']
```

Encarsia helps detect misconfigurations

DifuzzRTL

Bug	Detected	Module
5	✓	UOPCodeFDivDec ...
6	✗	DivUnit
7	✗	MulDiv
8	✓	IntToFP
9	✗	PipelinedMulUnit
10	✓	ALU
11	✓	RegisterReadDecode

Forgets the M extension

```
def __init__(self, isa='RV64G'):
    isas = []
    if 'I' in isa:
        isas += [ 'rv32i' ]
    if 'M' in isa:
        isas += [ 'rv32m' ]
    if 'A' in isa:
        isas += [ 'rv32a' ]
    if 'F' in isa:
        isas += [ 'rv32f' ]
    if 'G' in isa:
--      isas = ['rv32i', 'rv32m', 'rv32a', 'rv32f']
```

Insight 4:
Fuzzers do not test the entire ISA

Encarsia helps detect flaws

Cascade

Bug	Detected	Module
1	✓	IssueSlot
2	✓	Arbiter_16
3	✗	DivSqrtRecF64_mul ...
4	✗	DivSqrtRecF64_mul ...
5	✗	UOPCodeFDivDec ...
6	✓	DivUnit
7	✓	MulDiv
8		IntToFP
⋮	⋮	

Encarsia helps detect flaws

Cascade

Bug	Detected	Module
1	✓	IssueSlot
2	✓	Arbiter_16
3	✗	DivSqrtRecF64_mul ...
4	✗	DivSqrtRecF64_mul ...
5	✗	UOPCodeFDivDec ...
6	✓	DivUnit
7	✓	MulDiv
8		IntToFP
⋮		

Excludes whole instructions

```
if cpu == "cva6":
    probability["fsqrt.d"] = 0

    if not tolerate_cva6_division():
        probability["fdiv.d"] = 0

    if not tolerate_cva6_single_precision():
        probability["fsqrt.s"] = 0
        probability["fdiv.s"] = 0

if cpu == "boom":
++  probability["fsqrt.d"] = 0
++  probability["fdiv.d"] = 0
```

Encarsia helps detect flaws

Cascade

Bug	Detected	Module
1	✓	IssueSlot
2	✓	Arbiter_16
3	✗	DivSqrtRecF64_mul ...
4	✗	DivSqrtRecF64_mul ...
5	✗	UOPCodeFDivDec ...
6	✓	DivUnit
7	✓	MulDiv
8		IntToFP
⋮		⋮

Excludes whole instructions

```
if cpu == "cva6":
    probability["fsqrt.d"] = 0

    if not tolerate_cva6_division():
        probability["fdiv.d"] = 0

    if not tolerate_cva6_single_precision():
        probability["fsqrt.s"] = 0
        probability["fdiv.s"] = 0

if cpu == "boom":
++  probability["fsqrt.d"] = 0
++  probability["fdiv.d"] = 0
```

Insight 5:
Filtering mechanisms hinder the discovery of further bugs

Encarsia injects complex bugs

Encarsia injects complex bugs

BOOM Broken Conditional #3

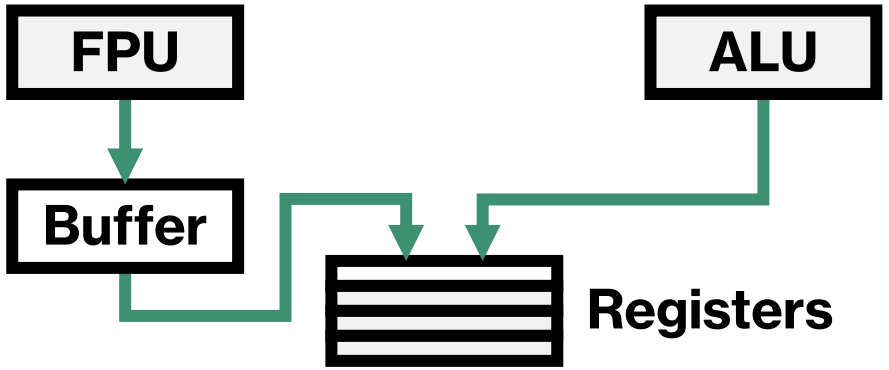
```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3’h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```

Encarsia injects complex bugs

BOOM Broken Conditional #3

```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3'h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```

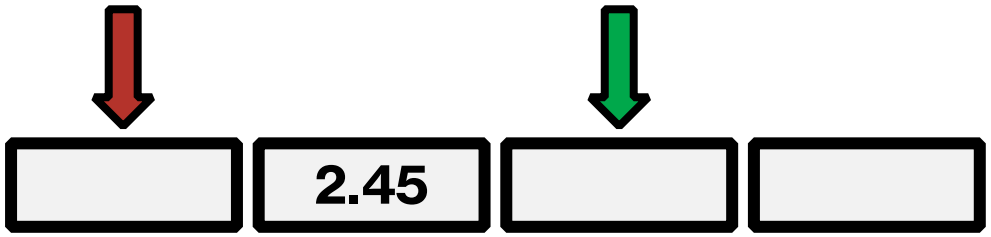
Buffer between FPU and GRs



Encarsia injects complex bugs

BOOM Broken Conditional #3

```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3'h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```

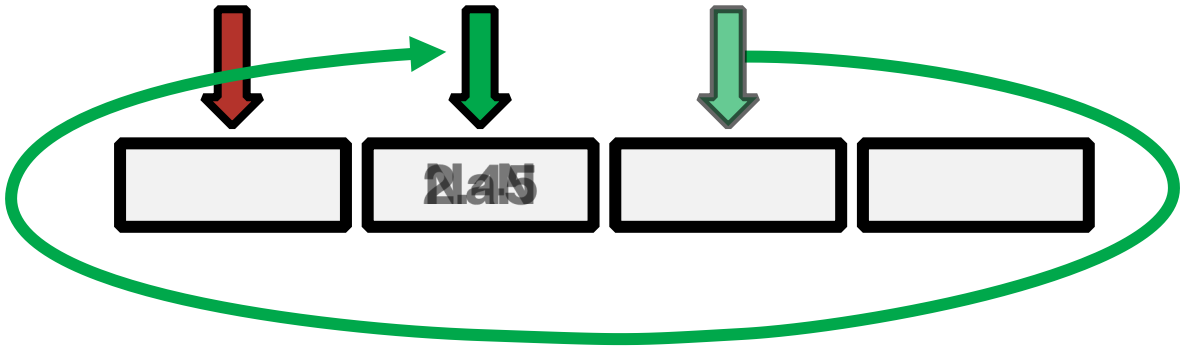


Buffer between FPU and GRs

Encarsia injects complex bugs

BOOM Broken Conditional #3

```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3'h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```

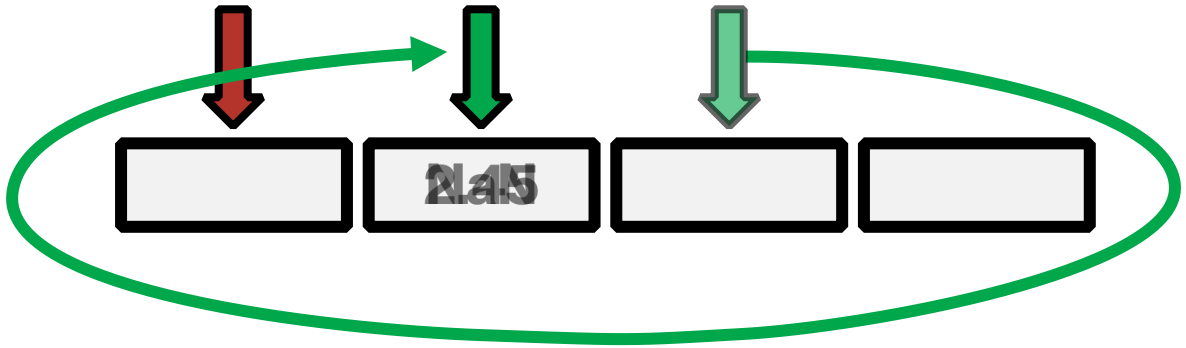


Buffer between FPU and GRs

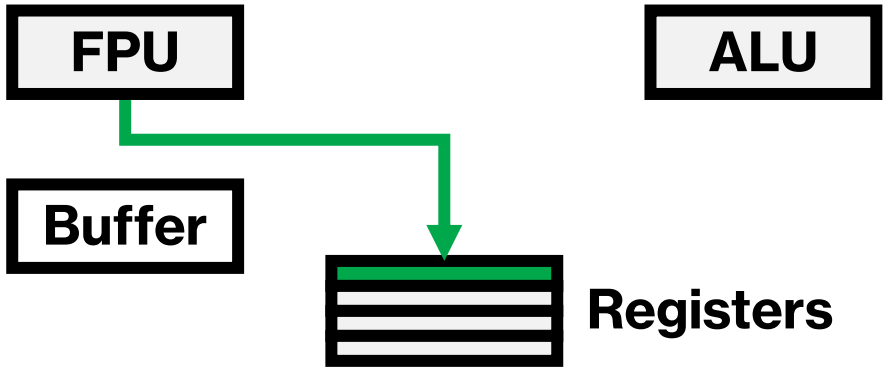
Encarsia injects complex bugs

BOOM Broken Conditional #3

```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3'h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```



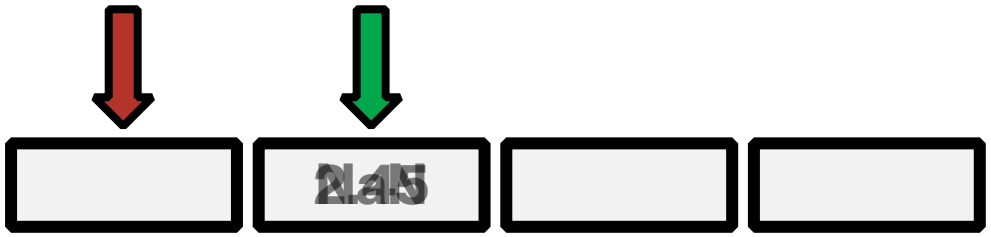
Buffer between FPU and GRs



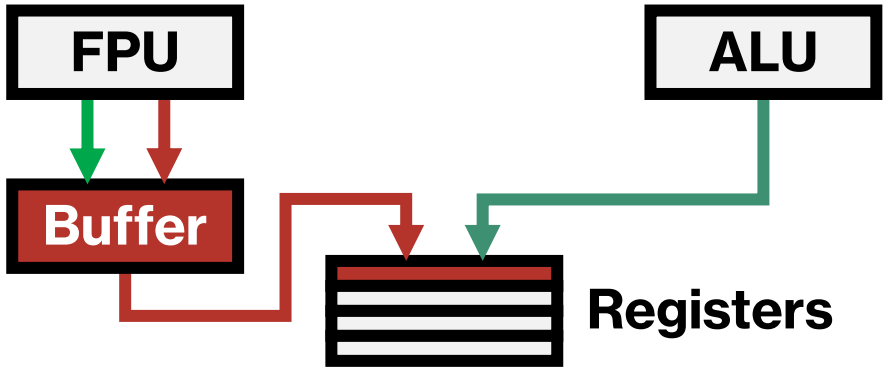
Encarsia injects complex bugs

BOOM Broken Conditional #3

```
if (reset) begin
    enq_ptr <= 3'h0;
end else if (do_enq) begin
    if (wrap) begin
        enq_ptr <= 3'h0;
++  end
end else begin
        enq_ptr <= enq_ptr + 1;
--  end
end
```



Buffer between FPU and GRs



Insight 6:
Fuzzers often miss subtle bugs from microarchitectural corner cases

Conclusion

- Hardware fuzzing is rife with false theories

Conclusion

- Hardware fuzzing is rife with false theories
- Accurate evaluation ensures transparency

Conclusion

- Hardware fuzzing is rife with false theories
- Accurate evaluation ensures transparency
- Reveals insights into improving future fuzzers