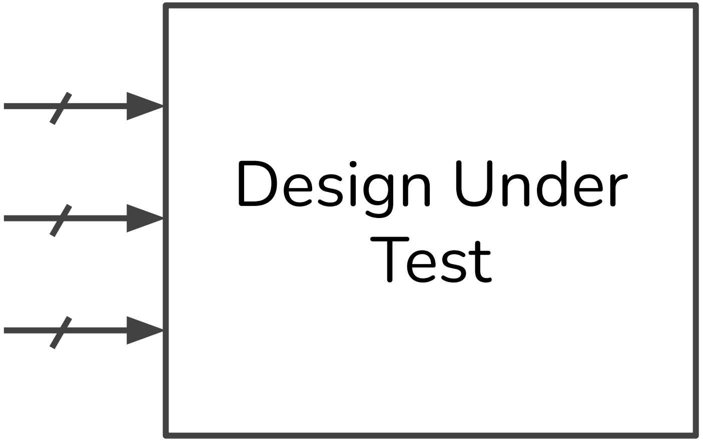
RFUZZ: Coverage-Directed Fuzz Testing of RTL on FPGAs

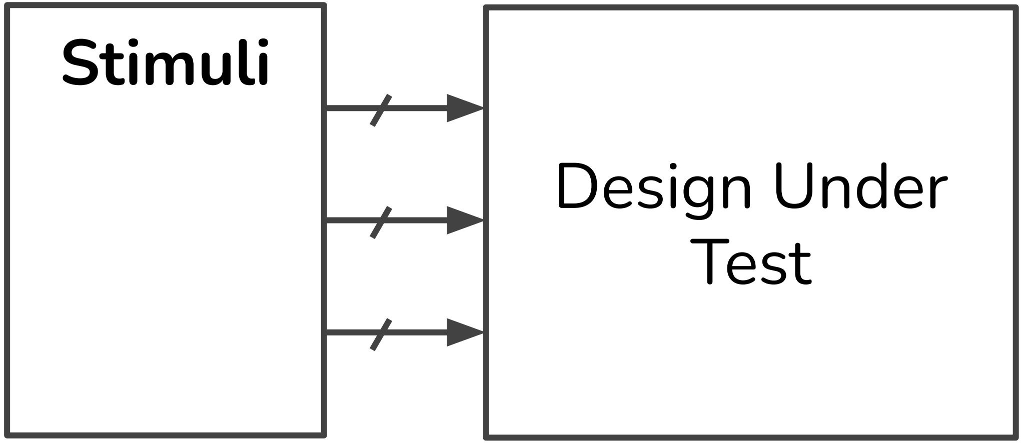
Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach and Koushik Sen
University of California, Berkeley
laeufer@cs.berkeley.edu

Dynamic Verification

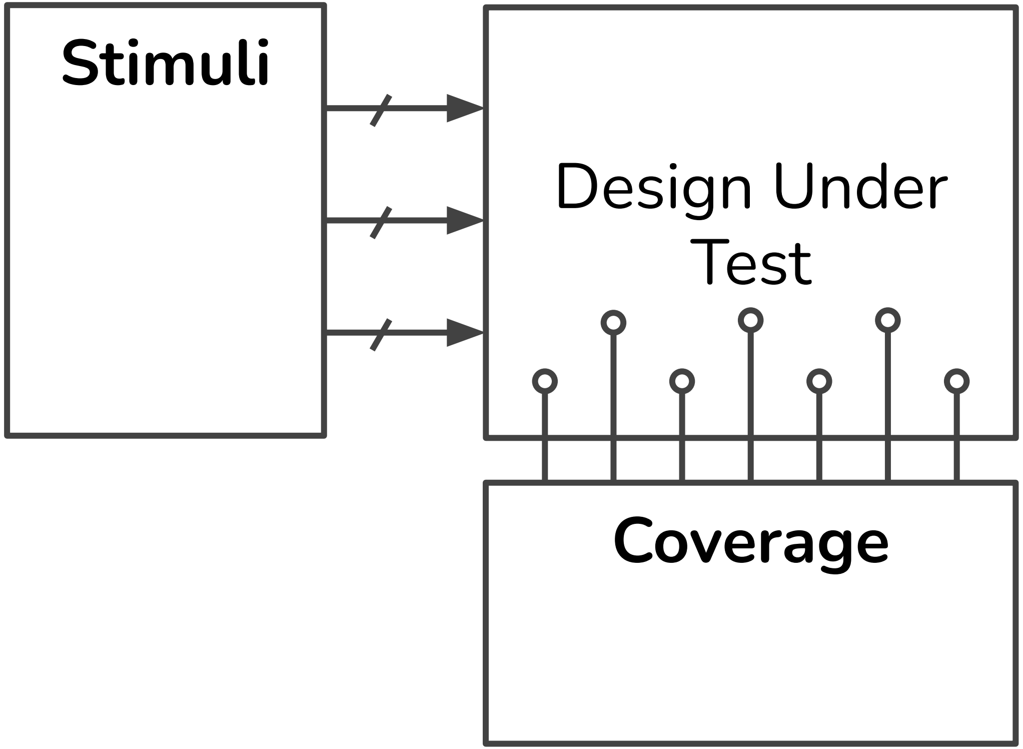
Dynamic Verification



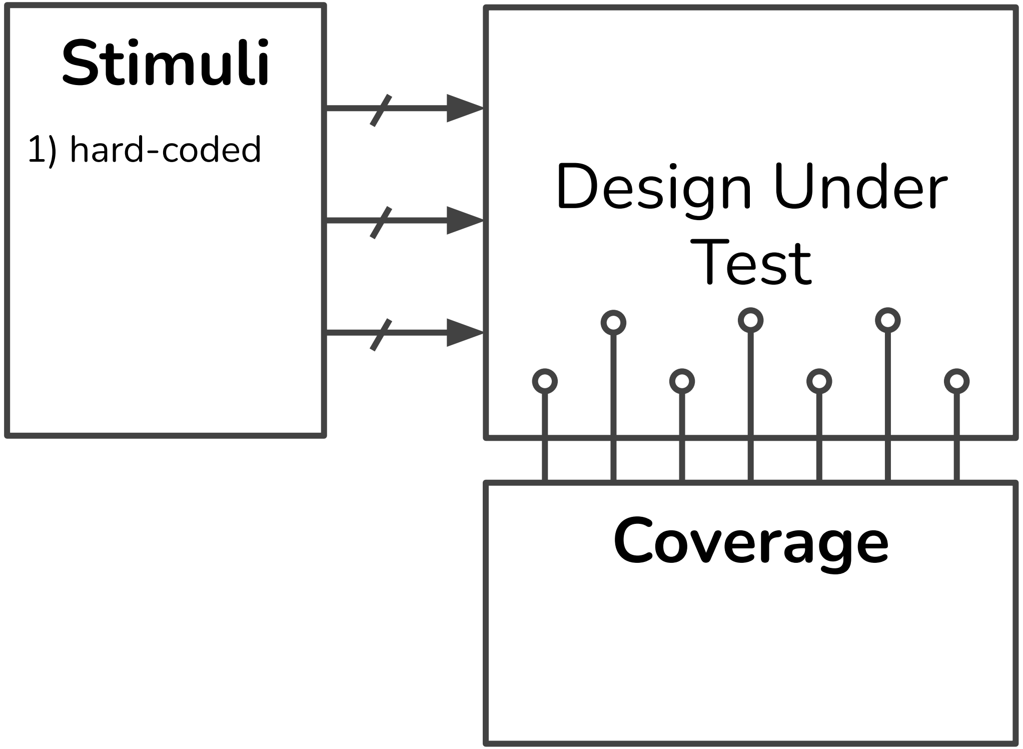
Dynamic Verification



Dynamic Verification



Dynamic Verification



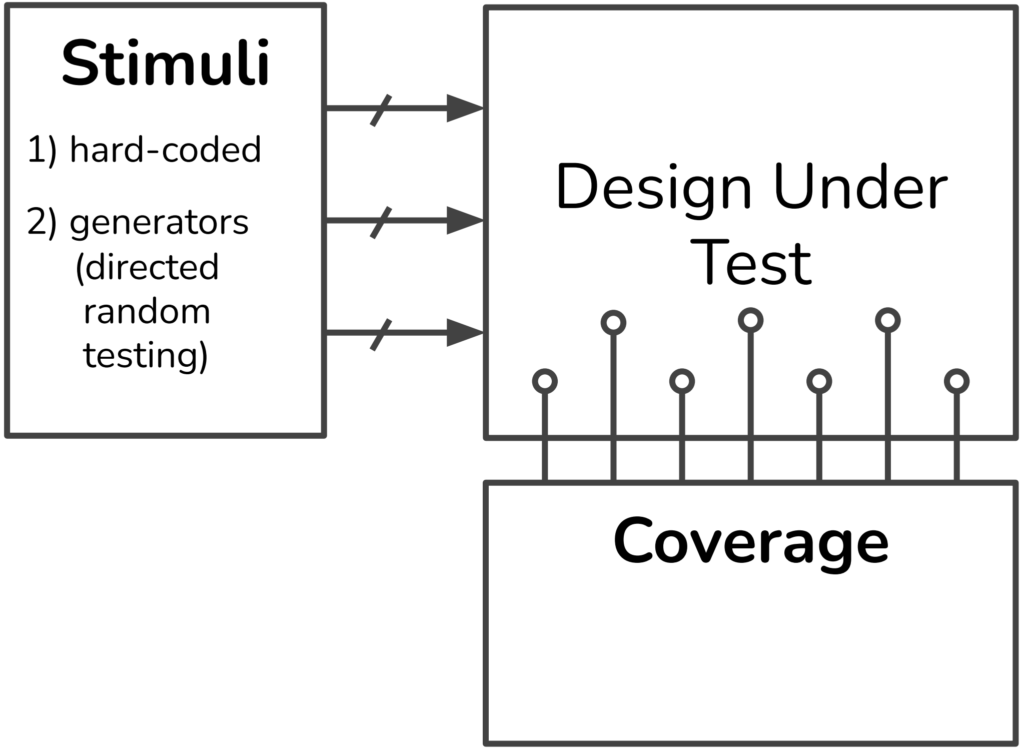
Stimuli

1) hard-coded

Design Under Test

Coverage

Dynamic Verification



Stimuli

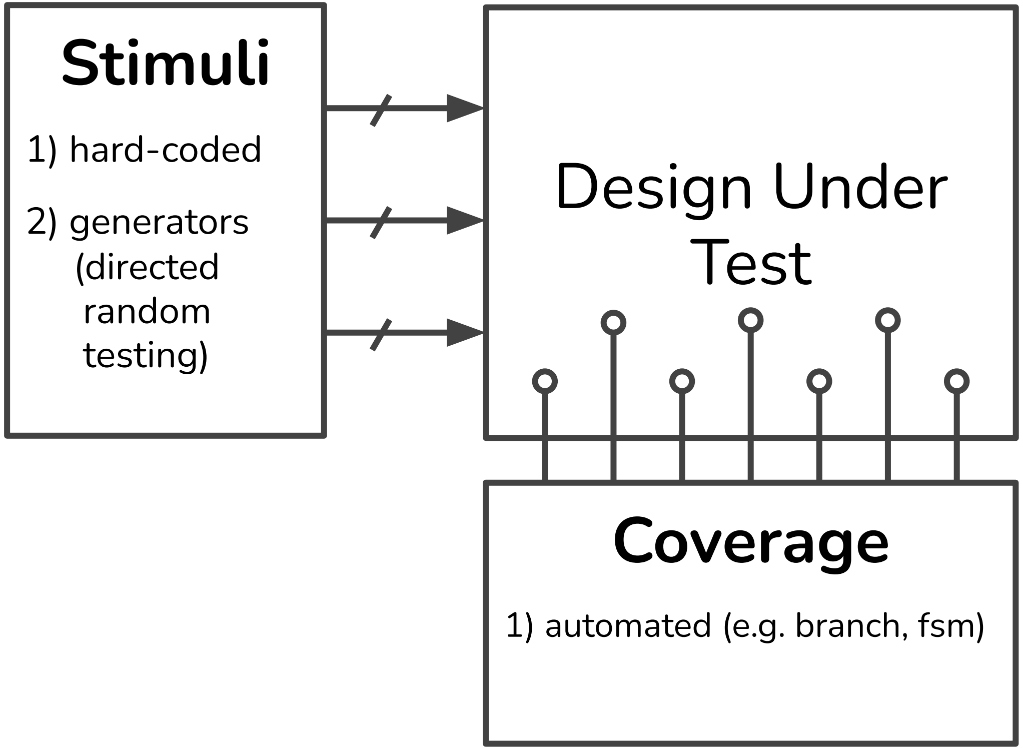
1) hard-coded

2) generators (directed random testing)

Design Under Test

Coverage

Dynamic Verification



Stimuli

1) hard-coded

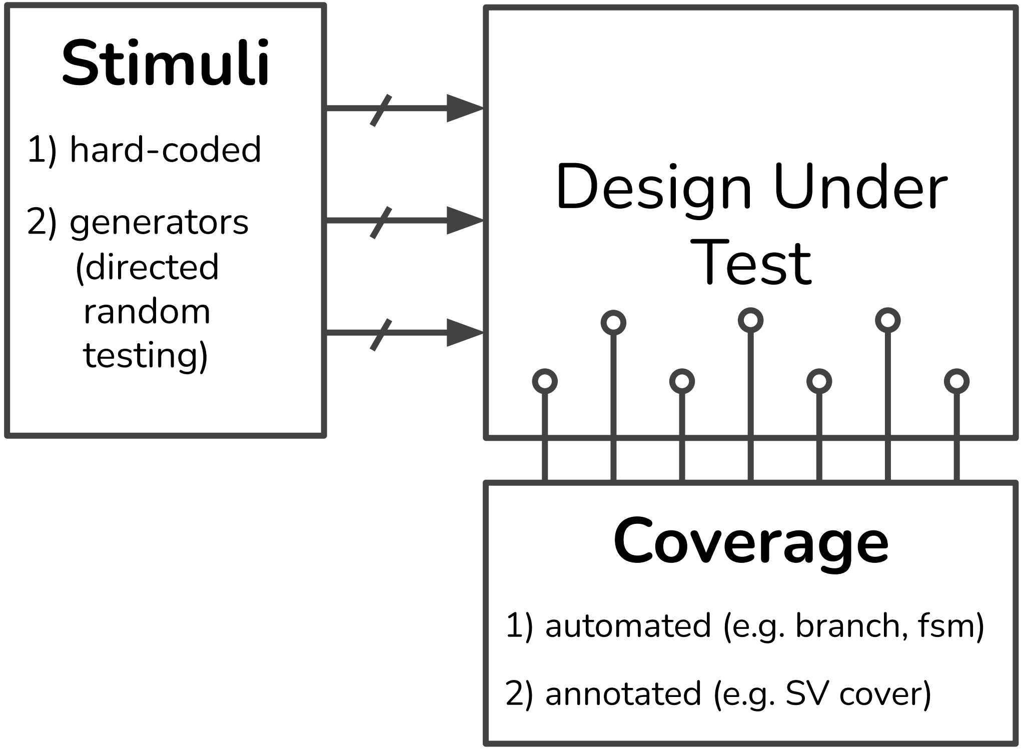
2) generators (directed random testing)

Design Under Test

Coverage

1) automated (e.g. branch, fsm)

Dynamic Verification



Stimuli

1) hard-coded

2) generators (directed random testing)

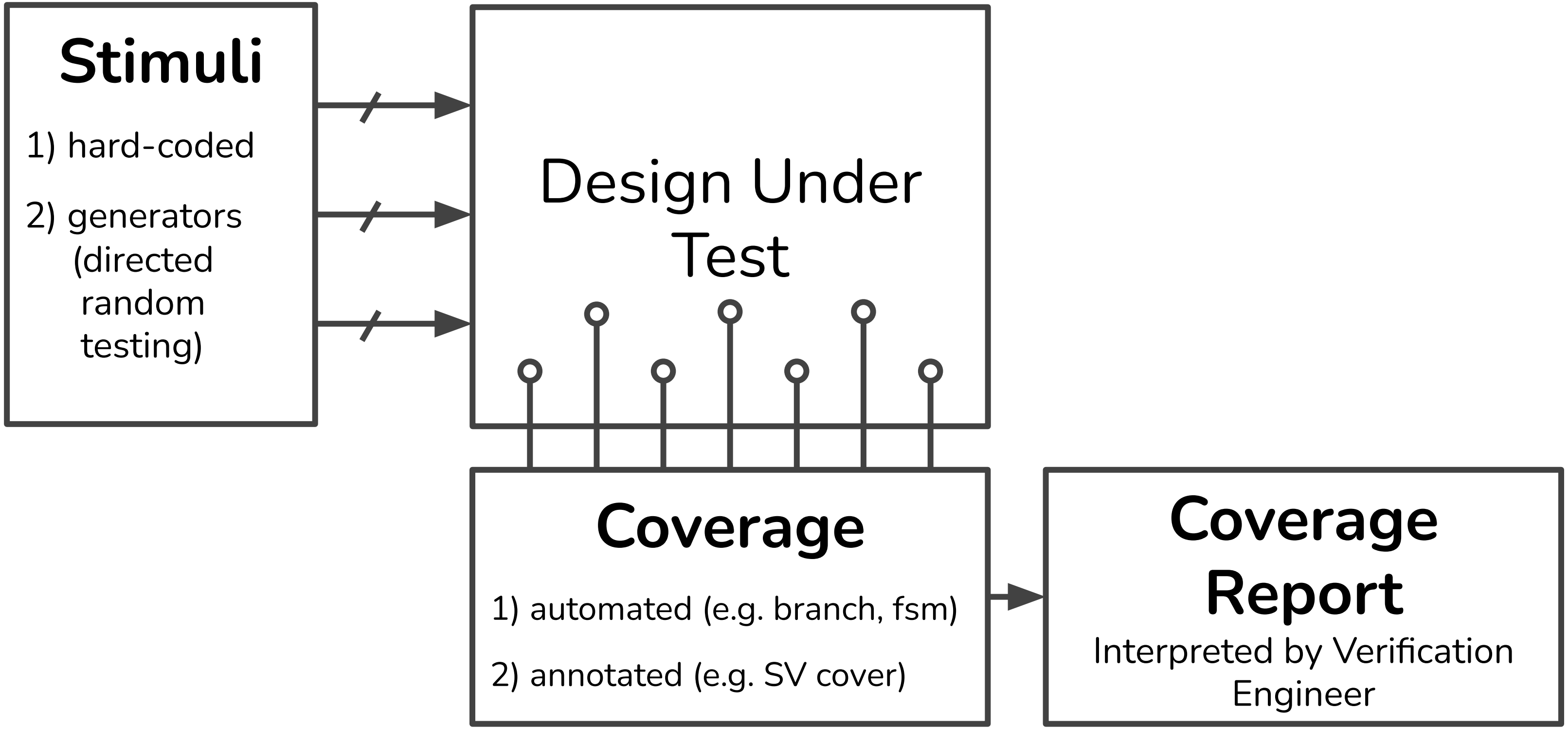
Design Under Test

Coverage

1) automated (e.g. branch, fsm)

2) annotated (e.g. SV cover)

Dynamic Verification



Stimuli

1) hard-coded

2) generators (directed random testing)

Design Under Test

Coverage

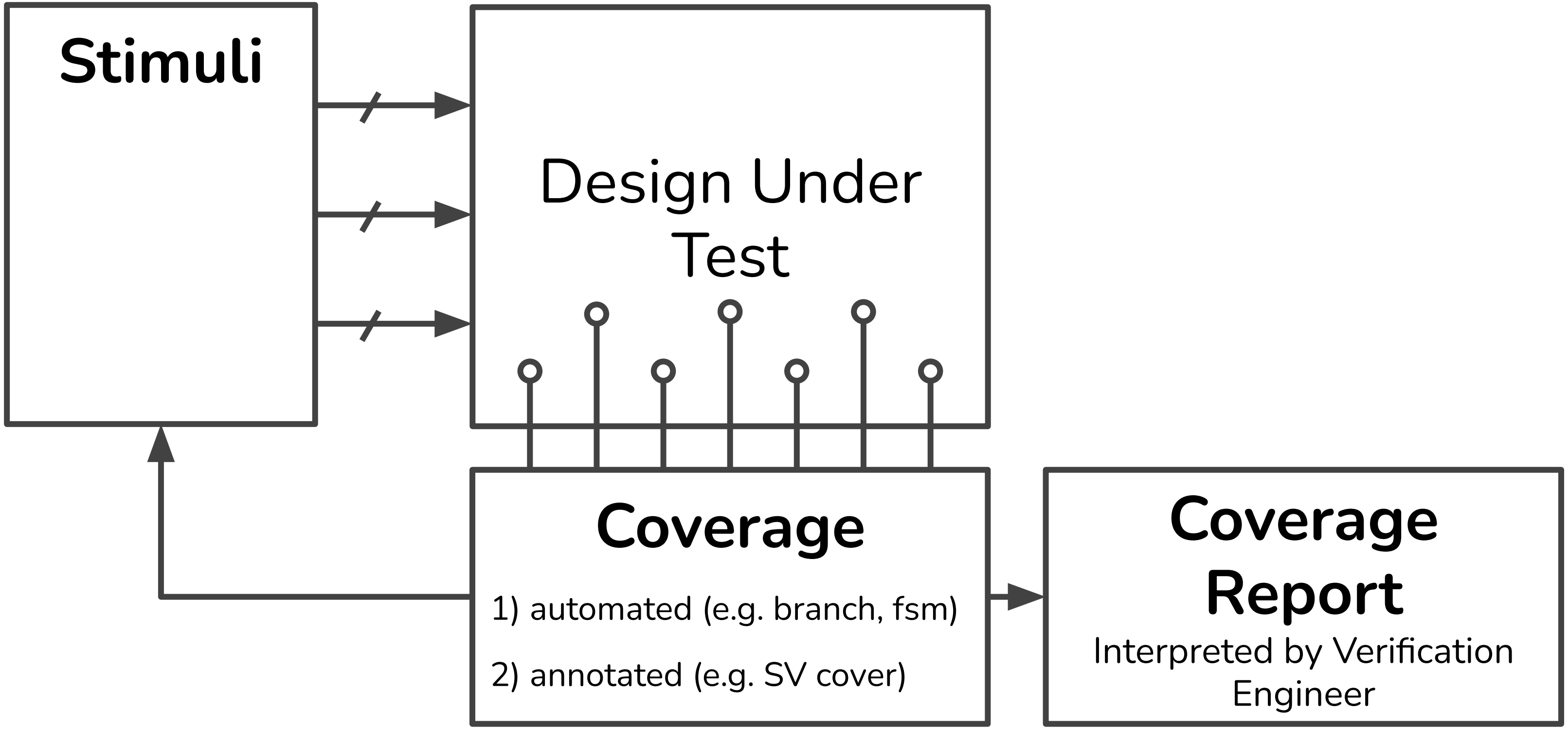
1) automated (e.g. branch, fsm)

2) annotated (e.g. SV cover)

Coverage Report

Interpreted by Verification Engineer

Dynamic Verification



Stimuli

Design Under Test

Coverage

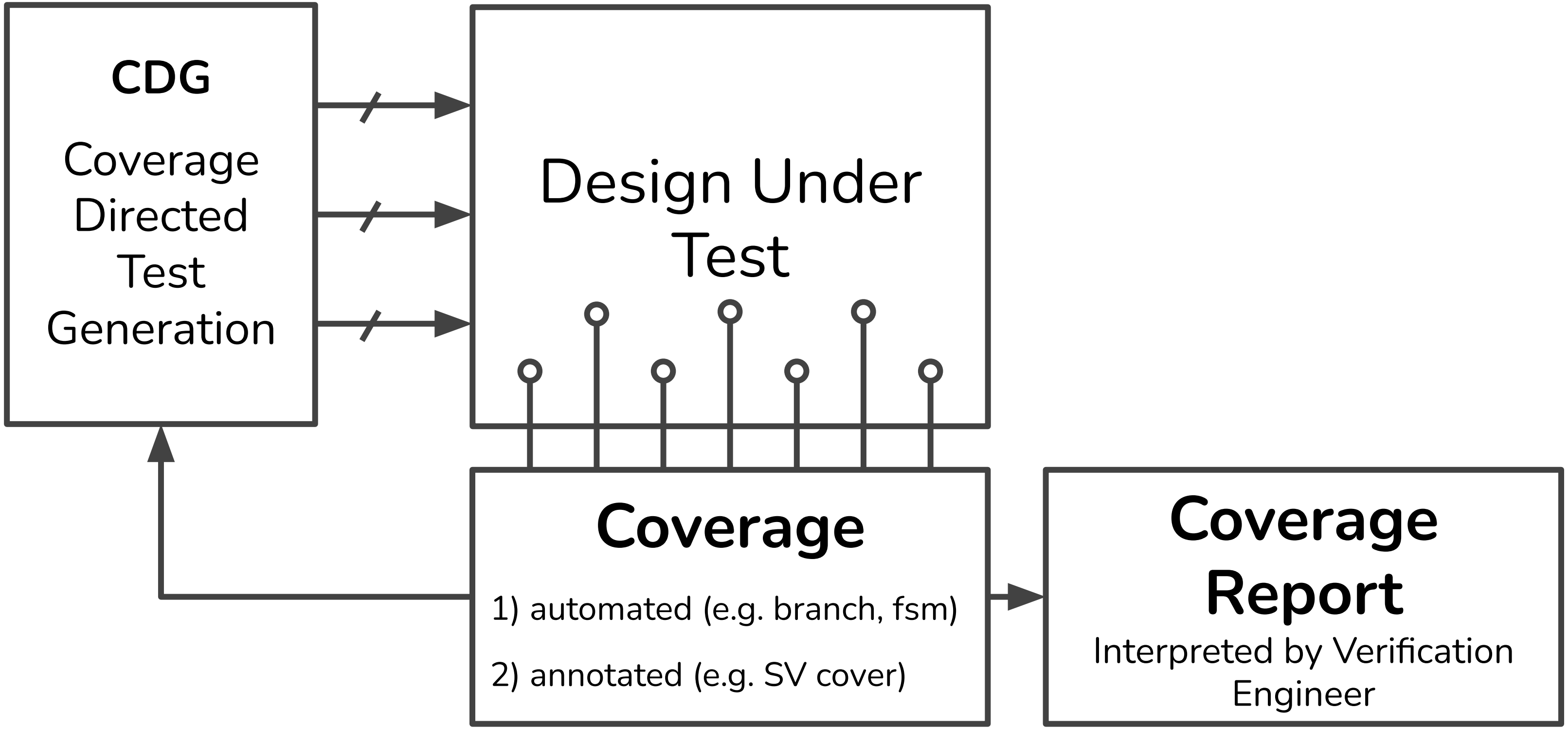
1) automated (e.g. branch, fsm)

2) annotated (e.g. SV cover)

Coverage Report

Interpreted by Verification Engineer

Dynamic Verification



CDG

Coverage Directed Test Generation

Design Under Test

Coverage

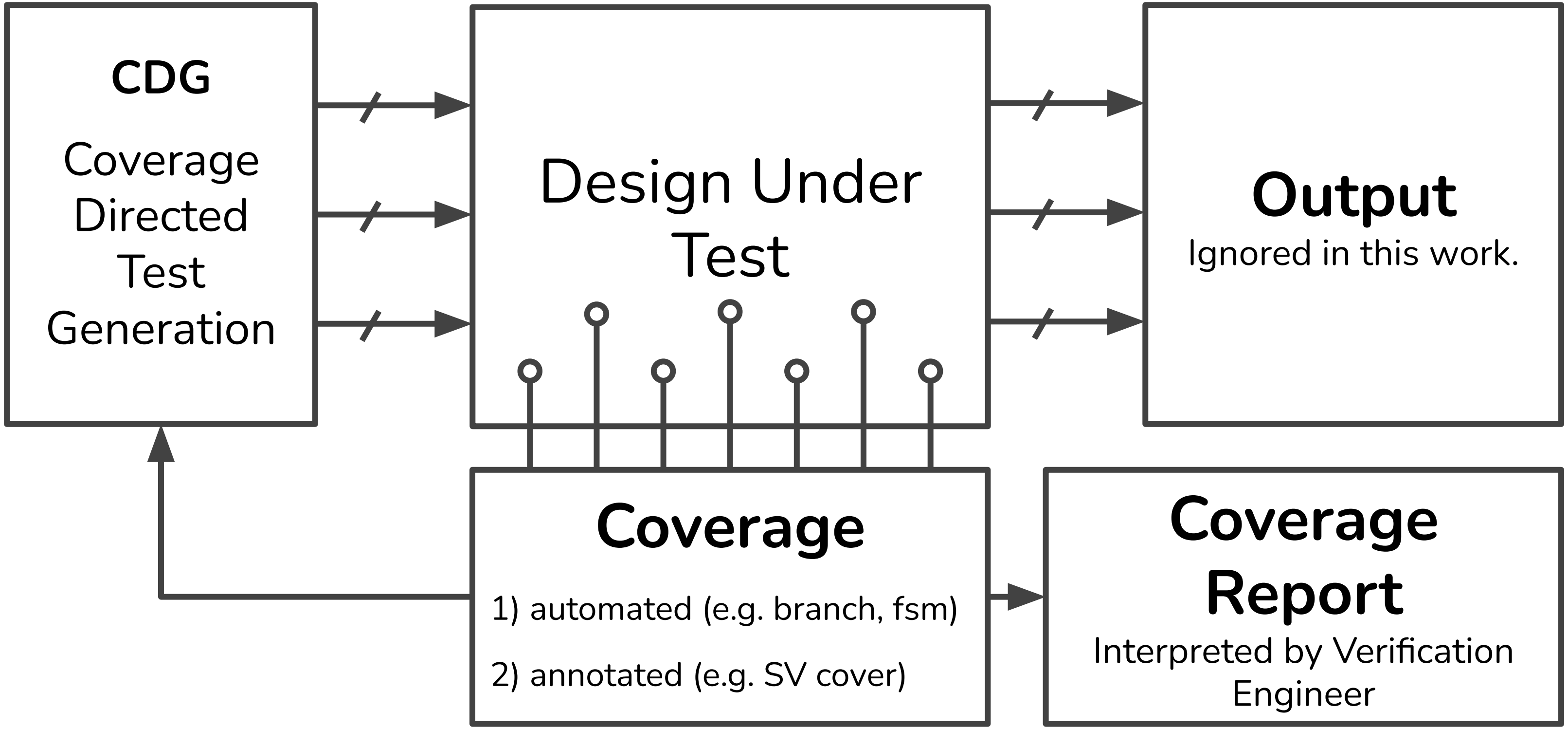
1) automated (e.g. branch, fsm)

2) annotated (e.g. SV cover)

Coverage Report

Interpreted by Verification Engineer

Dynamic Verification



CDG

Coverage Directed Test Generation

Design Under Test

Output

Ignored in this work.

Coverage

1) automated (e.g. branch, fsm)

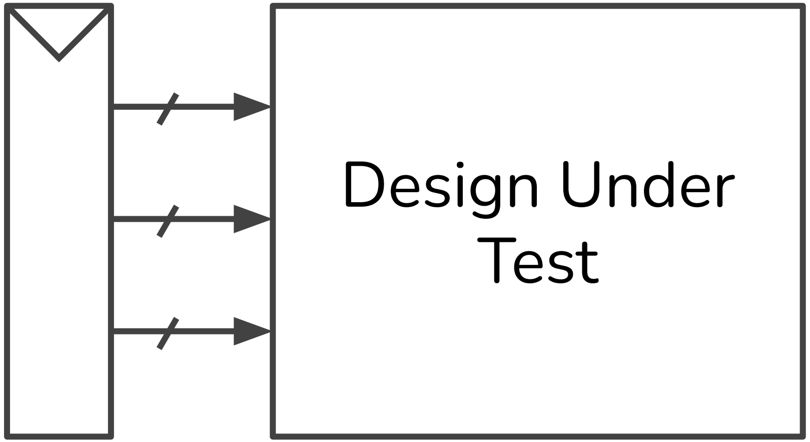
2) annotated (e.g. SV cover)

Coverage Report

Interpreted by Verification Engineer

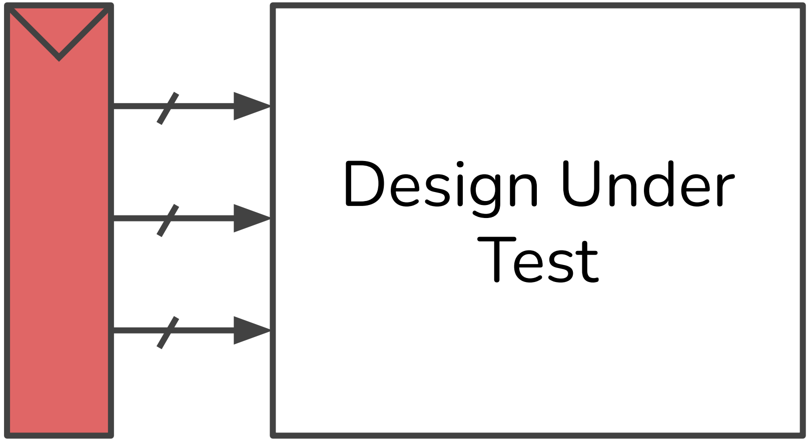
We use Fuzz Testing to generate stimuli from coverage feedback

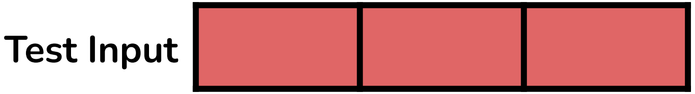
Input Definition



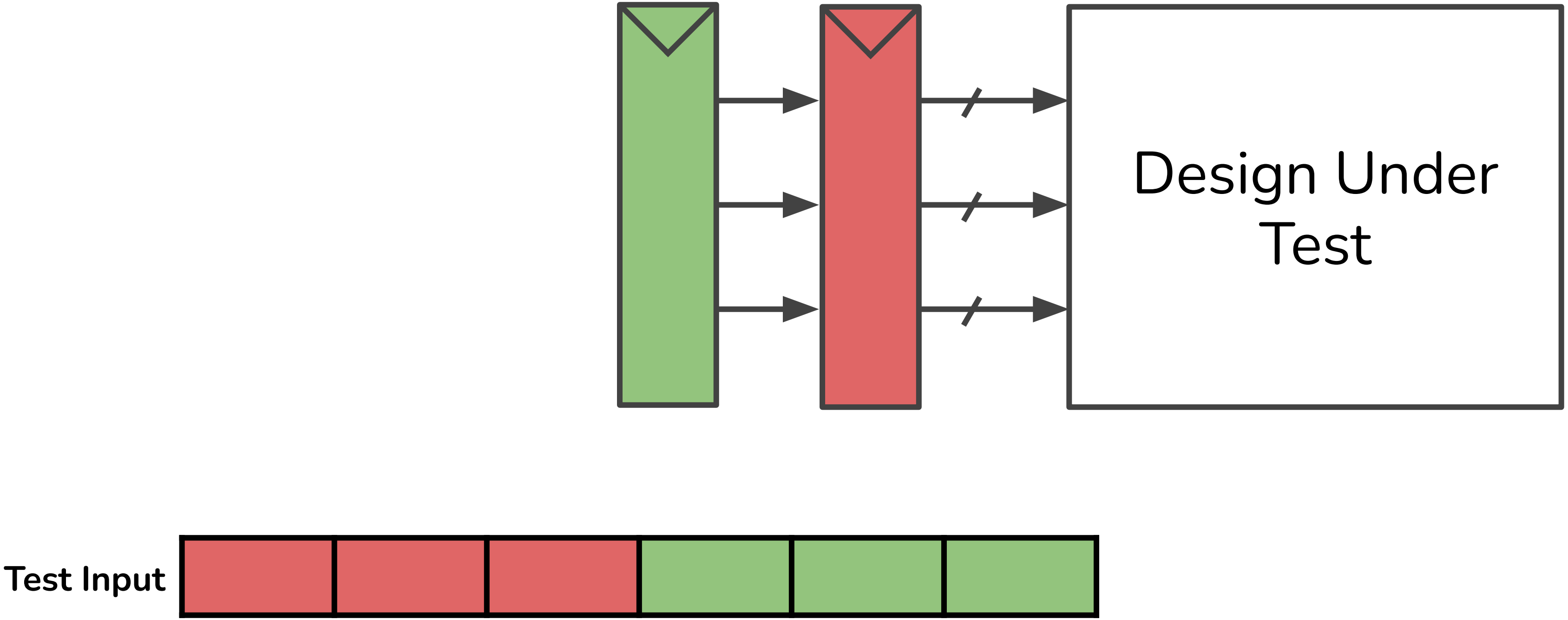
Test Input

Input Definition

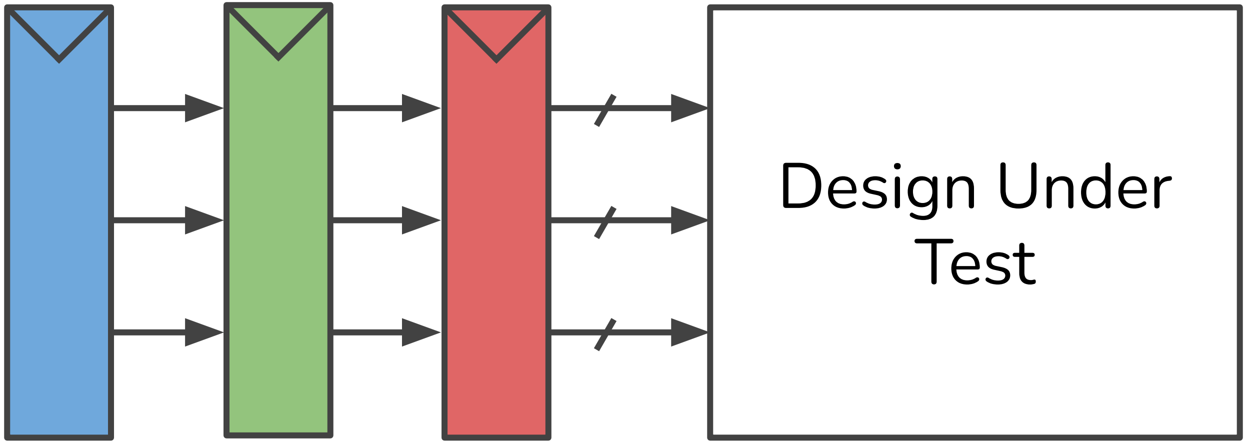


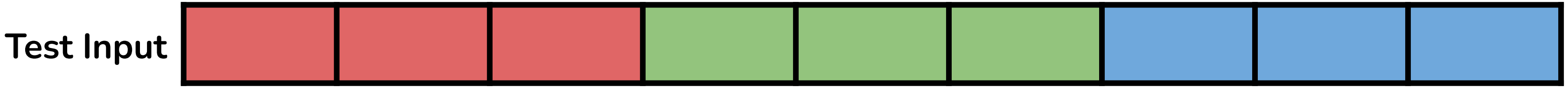


Input Definition



Input Definition





Coverage Definition

Functional Coverage

Coverage Definition

Functional Coverage

based on developer intent

Coverage Definition

Functional Coverage

based on developer intent

not available for open source designs

Coverage Definition

Functional Coverage

Automatic Coverage

based on developer intent

not available for open source designs

Coverage Definition

Functional Coverage

based on developer intent

not available for open source designs

Automatic Coverage

used to track test quality in absence of functional coverage

Coverage Definition

Functional Coverage

based on developer intent

not available for open source designs

Automatic Coverage

used to track test quality in absence of functional coverage

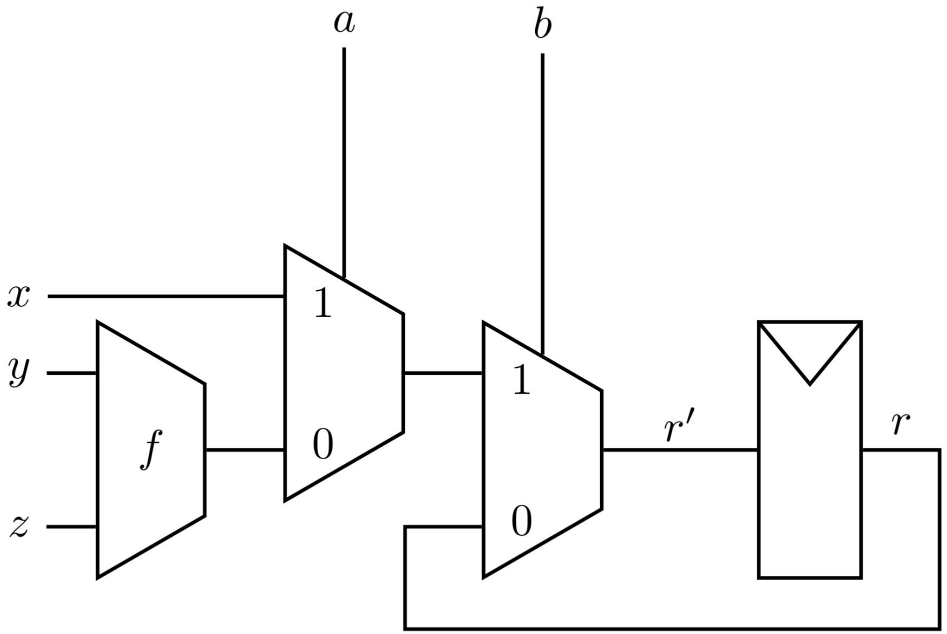
normally derived from HDL source, not RTL

Coverage Definition

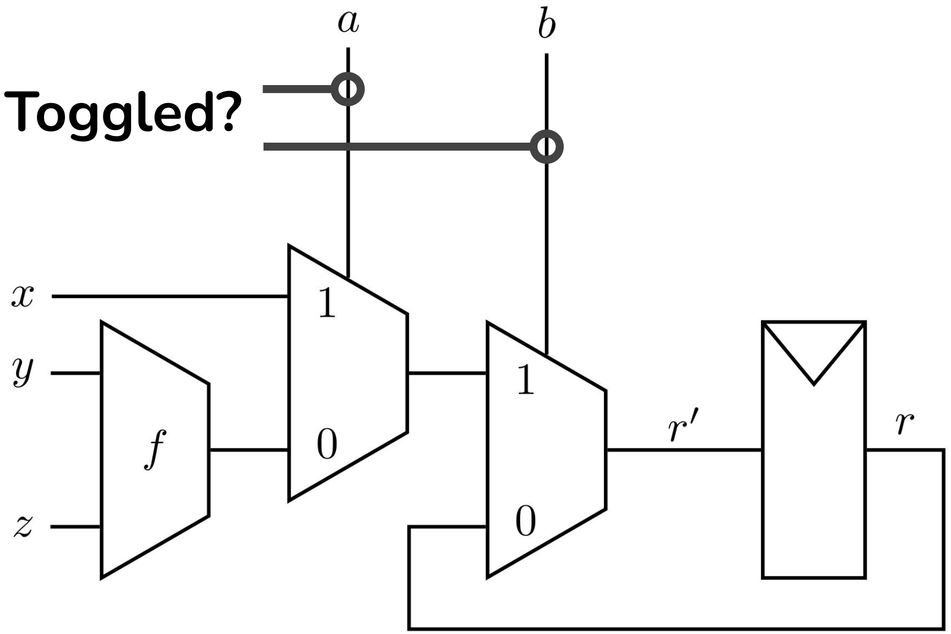
Functional Coverage	Automatic Coverage
based on developer intent	used to track test quality in absence of functional coverage
not available for open source designs	normally derived from HDL source, not RTL

→ we need an **automatic** coverage metric **based on RTL** netlist

Mux (Control) Toggle Coverage

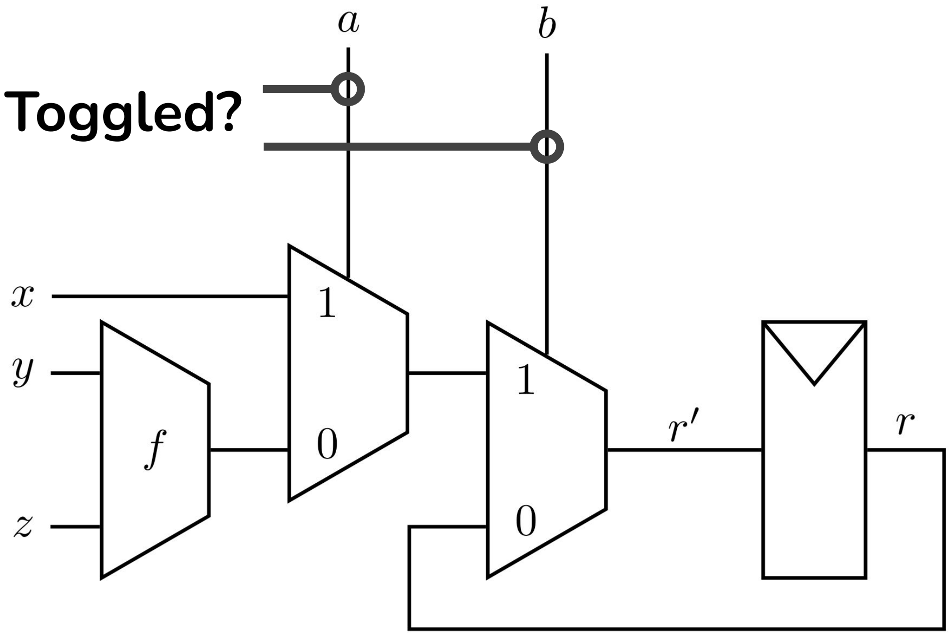


Mux Toggle Coverage



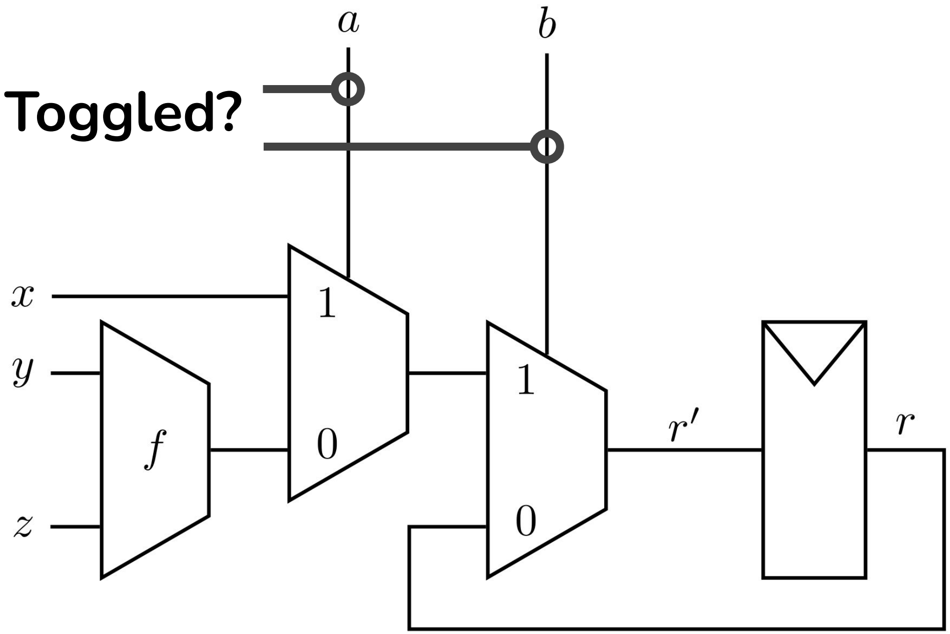
Mux Toggle Coverage

```
always @(posedge clk)
begin
  if (a) begin
    a_out = x;
  end else begin
    a_out = f(y,z);
  end
  if (b) begin
    r <= a_out;
  end
end
```



Toggled?

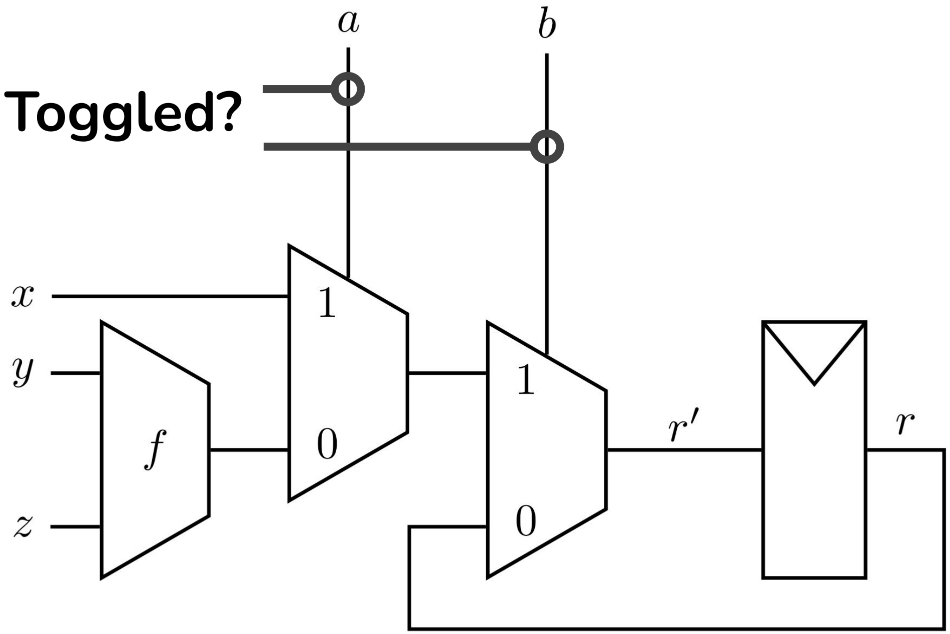
Mux Toggle Coverage


Toggled?

```
always @(posedge clk)
begin
  if (a) begin
     a_out = x;
  end else begin
     a_out = f(y,z);
  end
  if (b) begin
     r <= a_out;
  end
end
```

```
always @(posedge clk)
begin
  if (b) begin
    if (a) begin
      r <= x;
    end else begin
      r <= f(y, z);
    end
  end
end
```

Mux Toggle Coverage

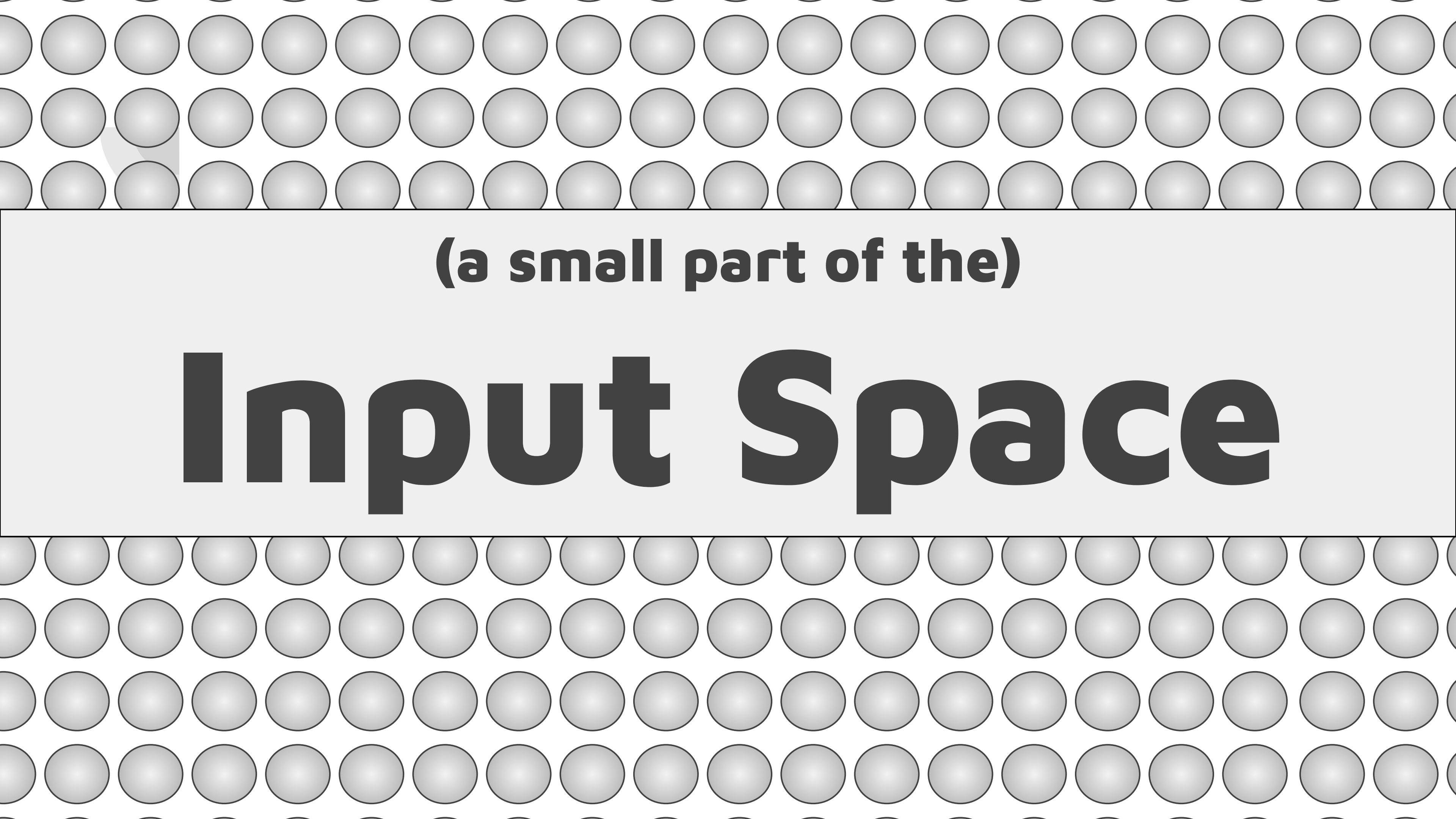


```
always @(posedge clk)
begin
  if (a) begin
     a_out = x;
  end else begin
     a_out = f(y,z);
  end
  if (b) begin
     r <= a_out;
  end
end
```

```
always @(posedge clk)
begin
  if (b) begin
    if (a) begin
      r <= x;
    end else begin
      r <= f(y, z);
    end
  end
end
```

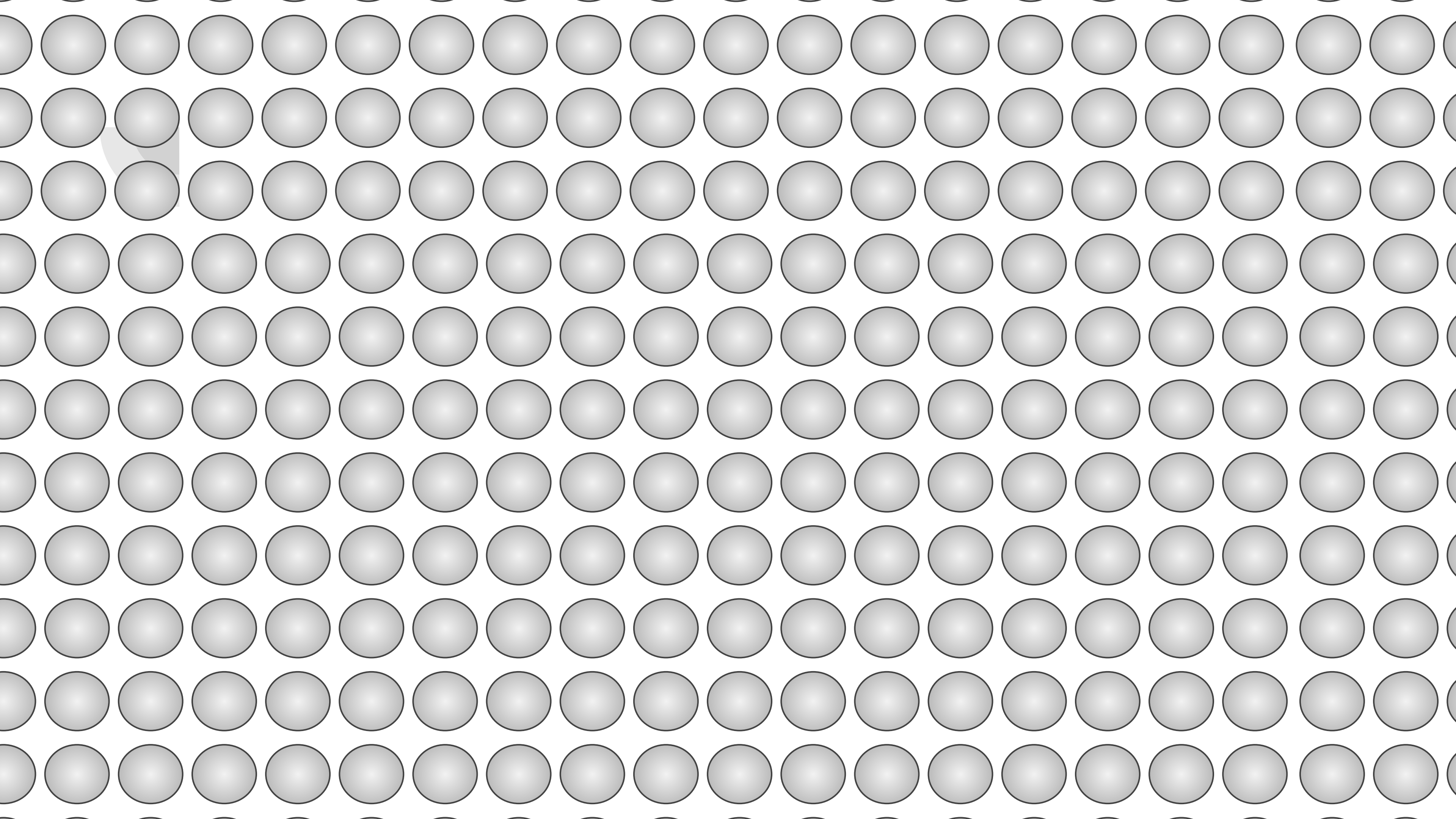
Background:

Coverage-Directed Fuzzing

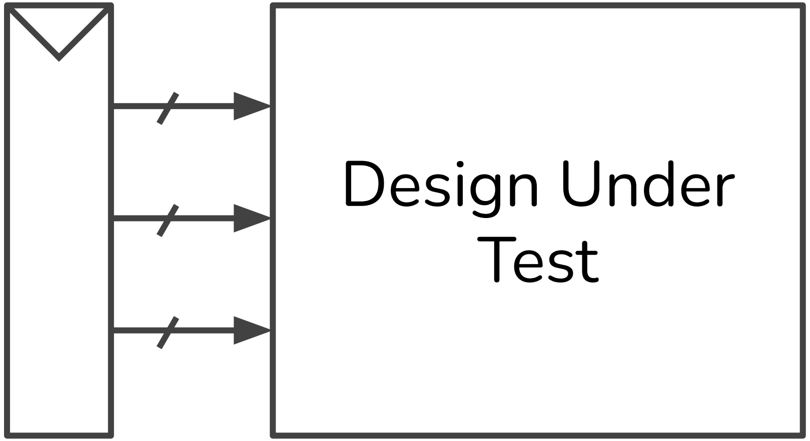


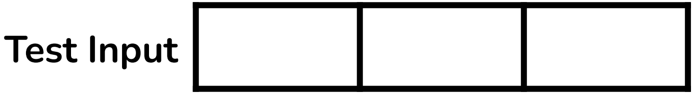
(a small part of the)

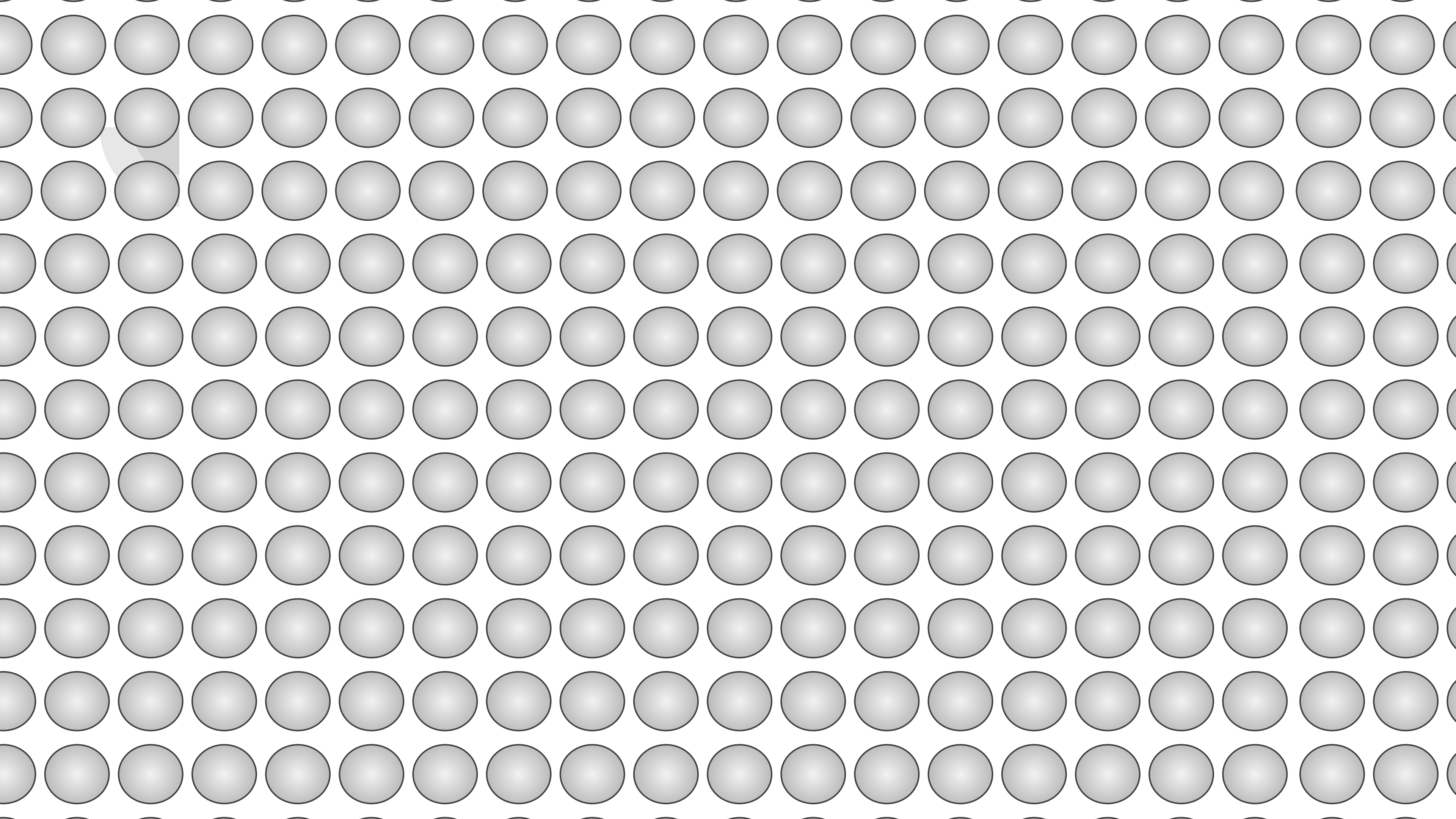
Input Space

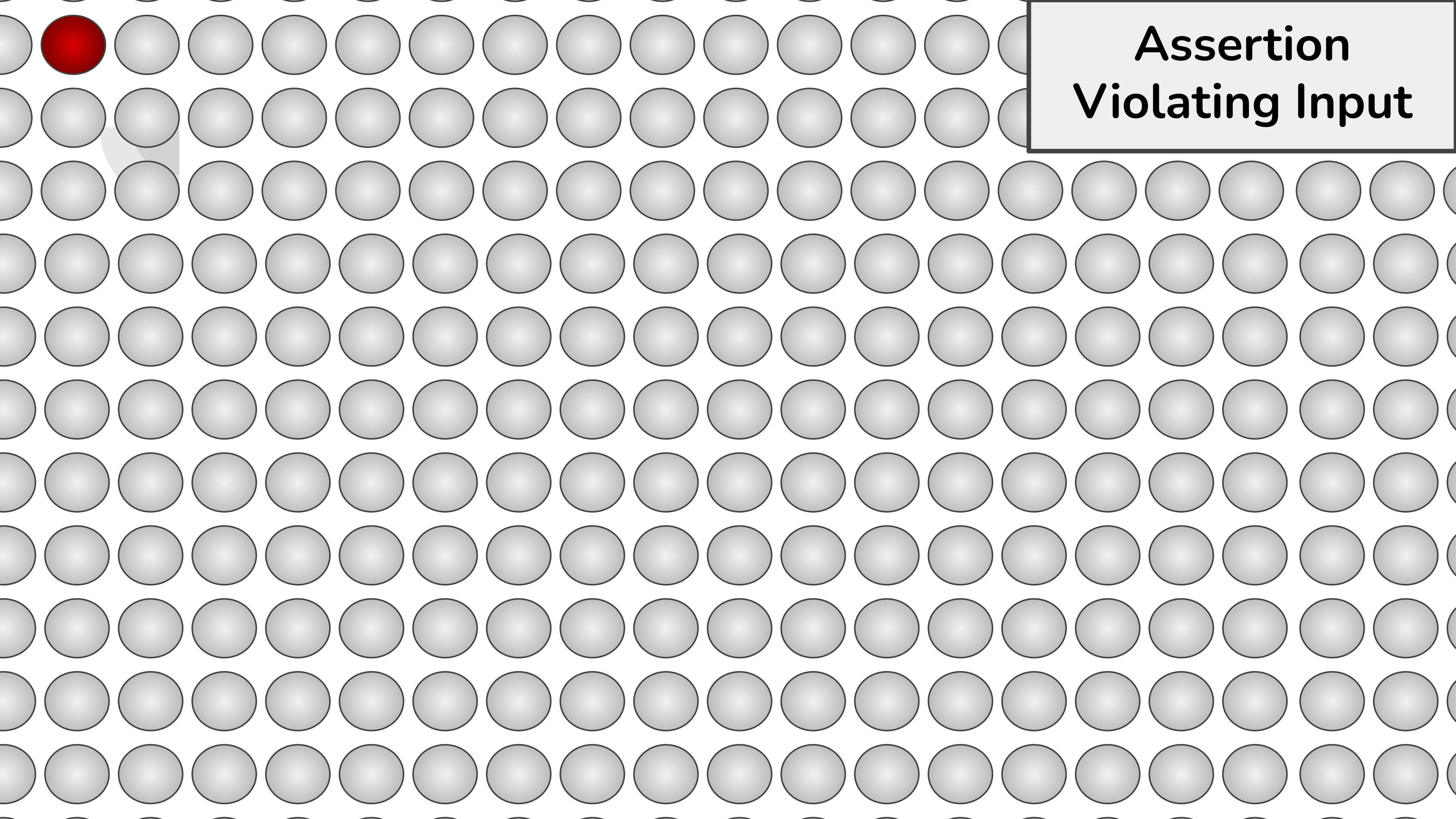


Input Definition

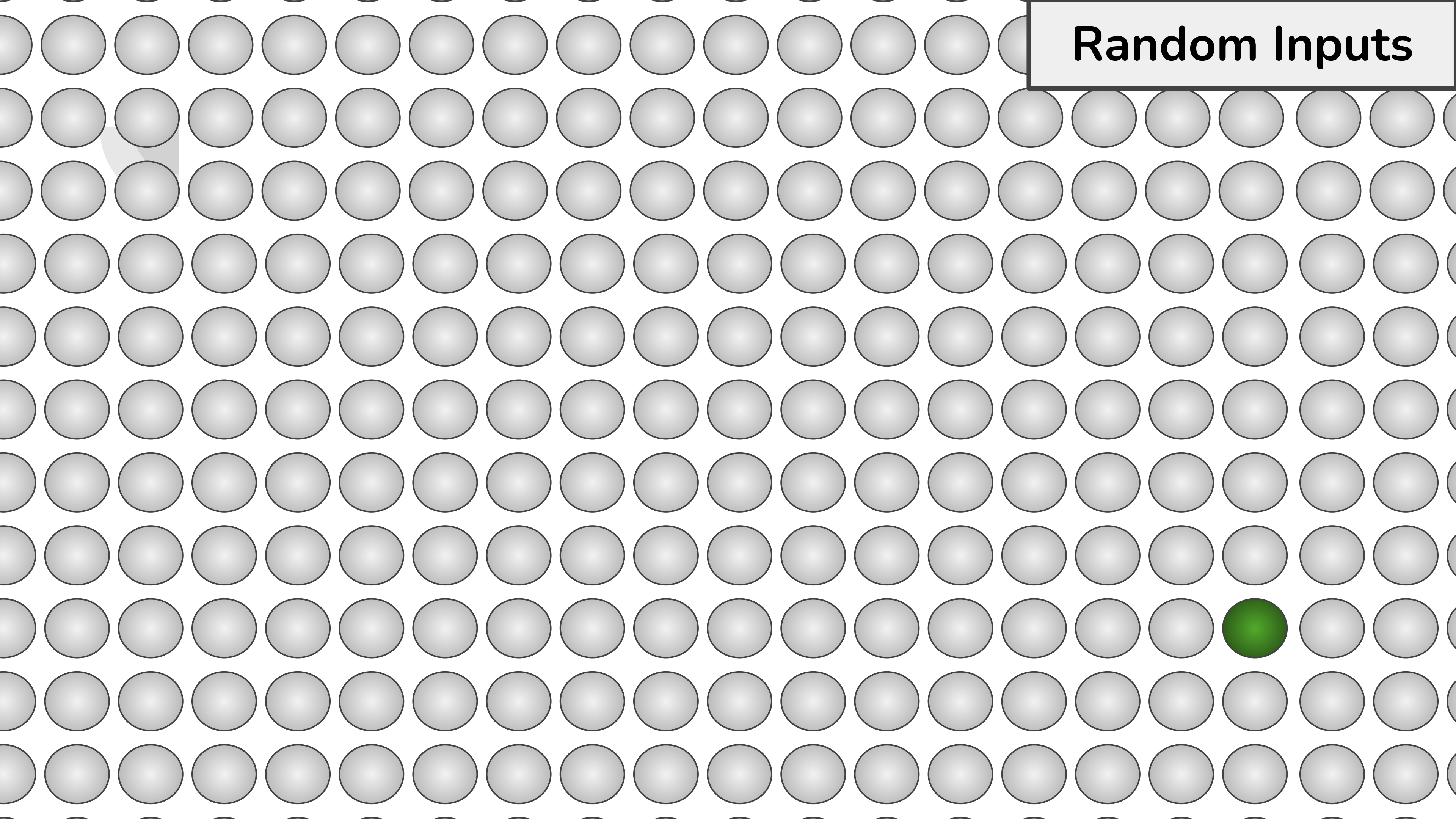




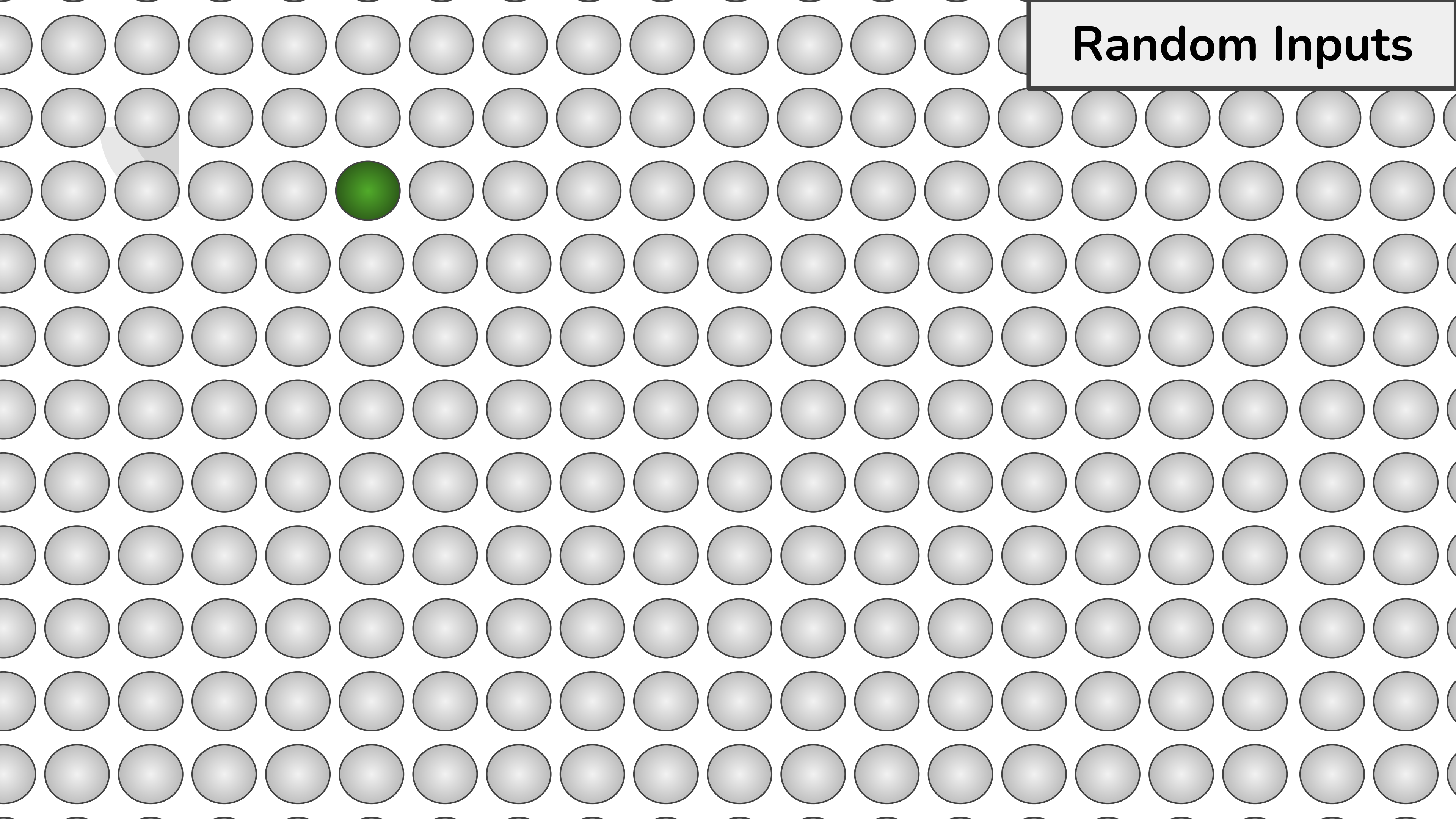




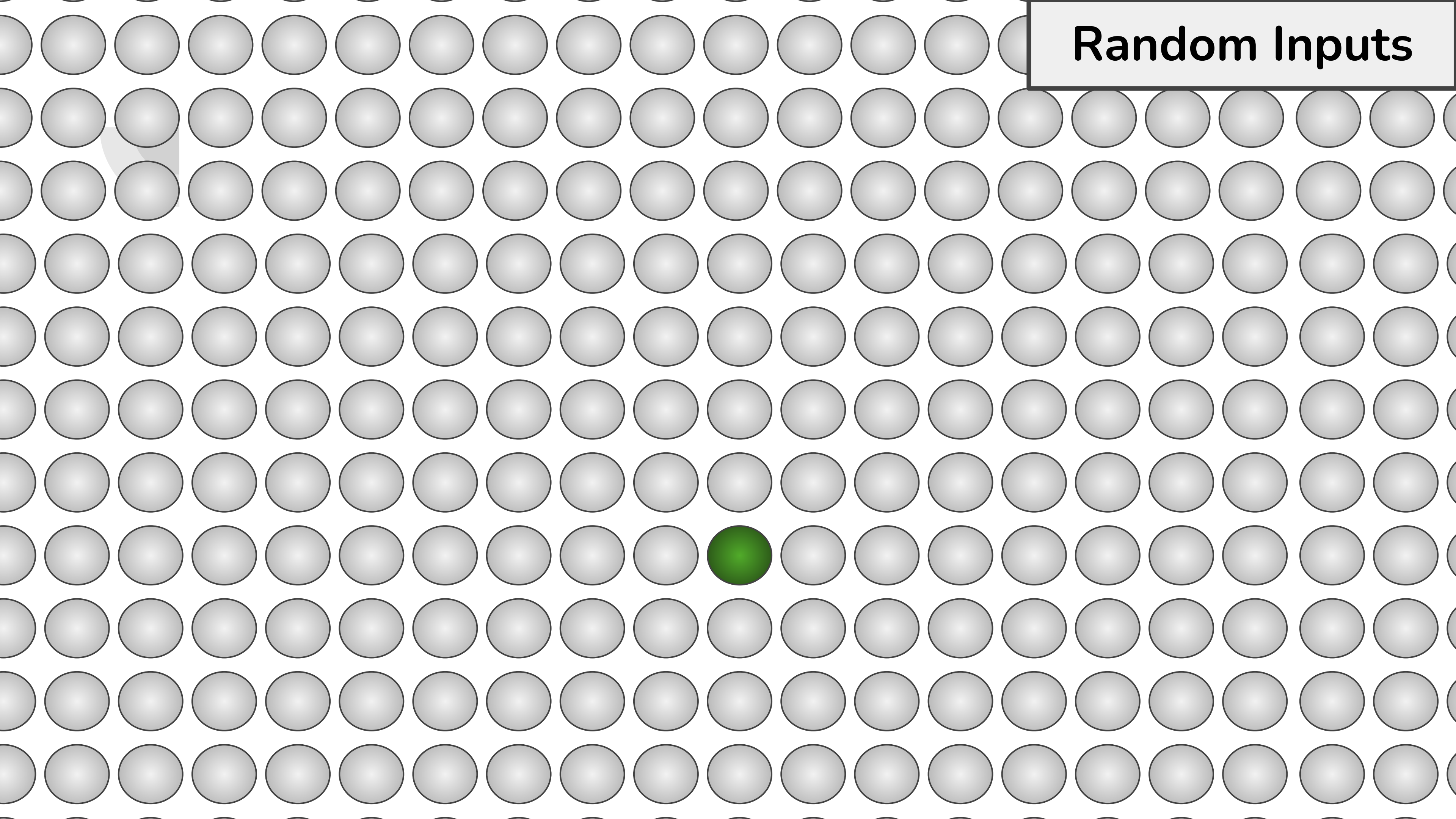
Assertion Violating Input



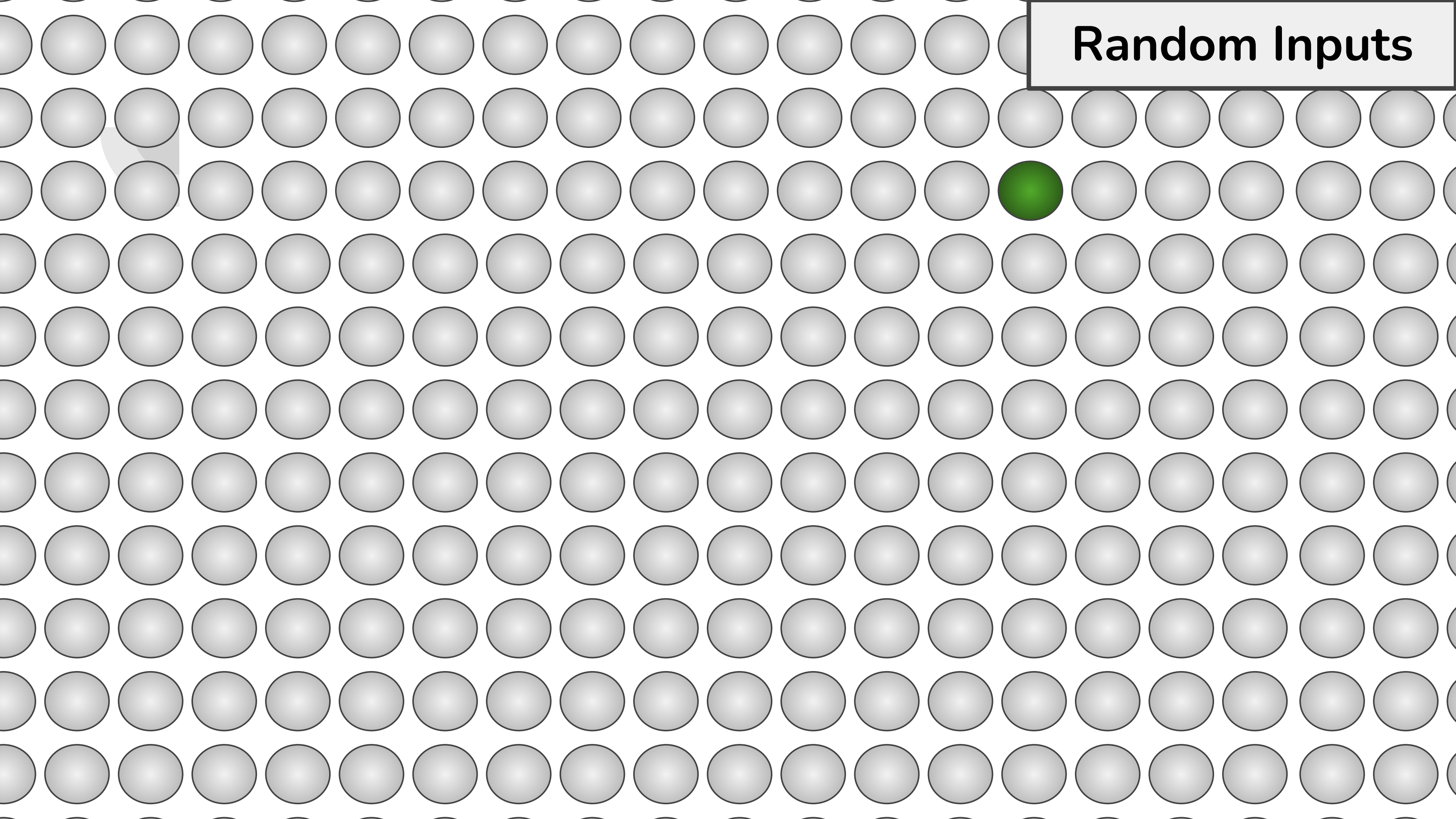
Random Inputs



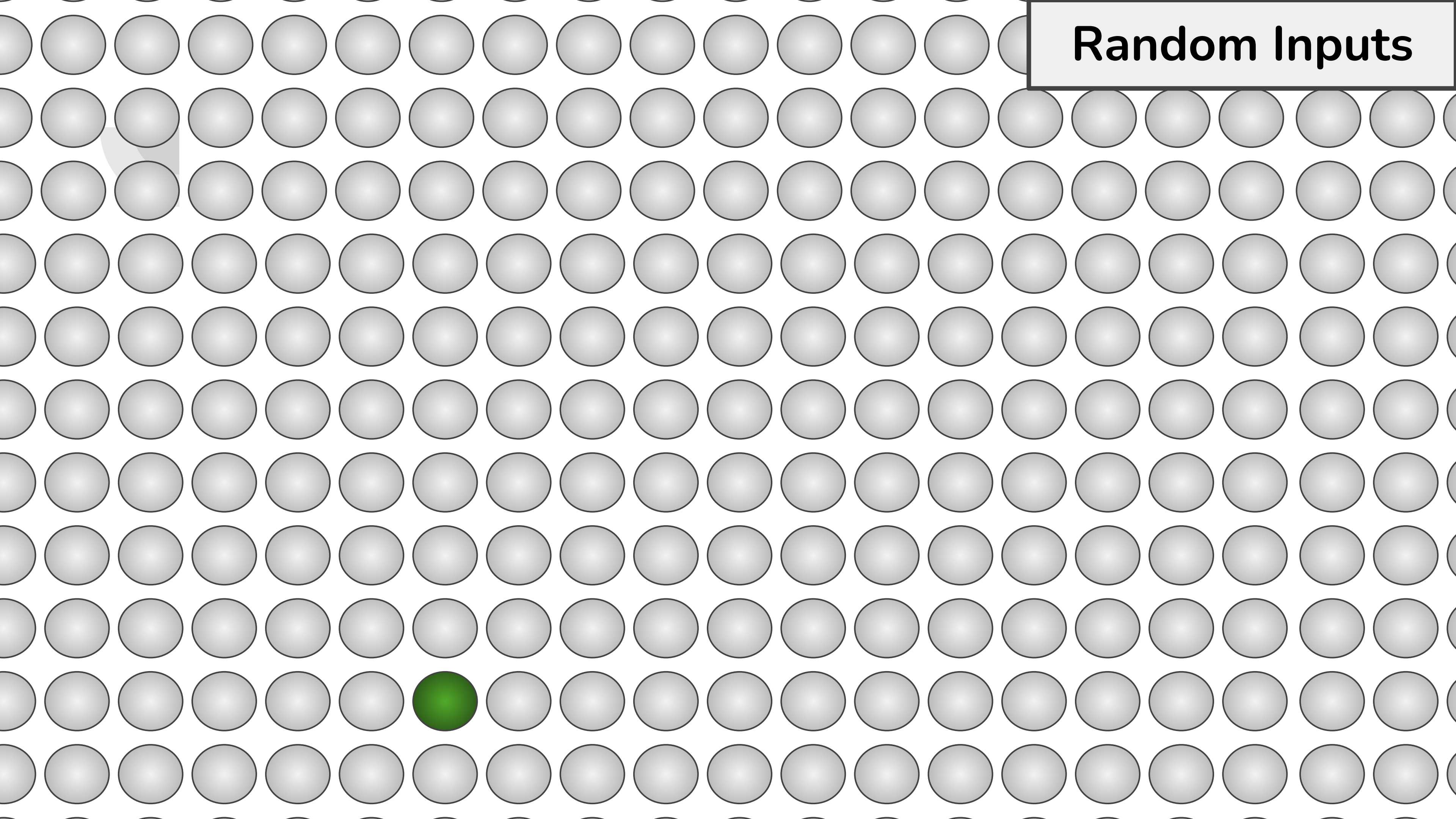
Random Inputs



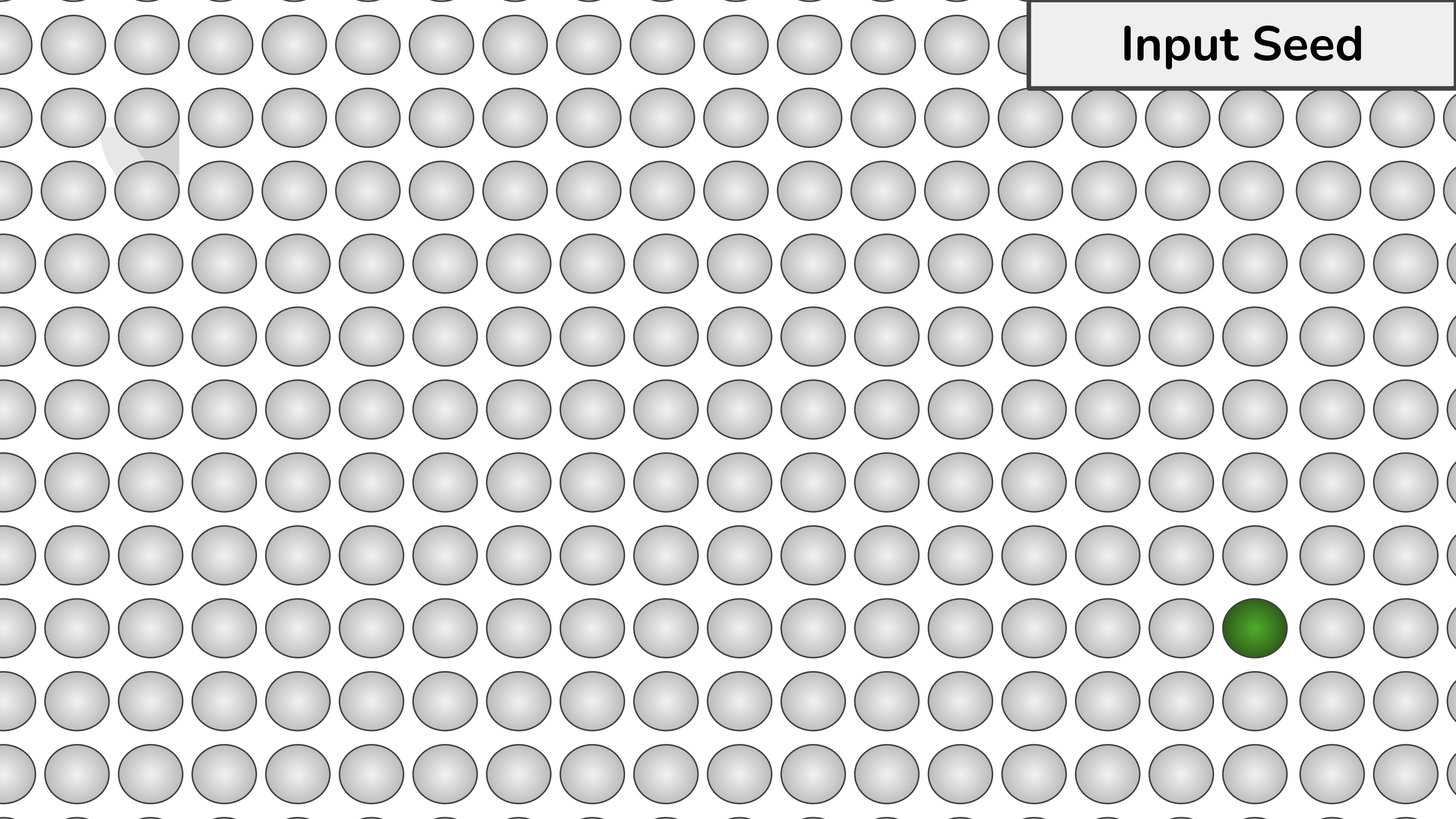
Random Inputs



Random Inputs

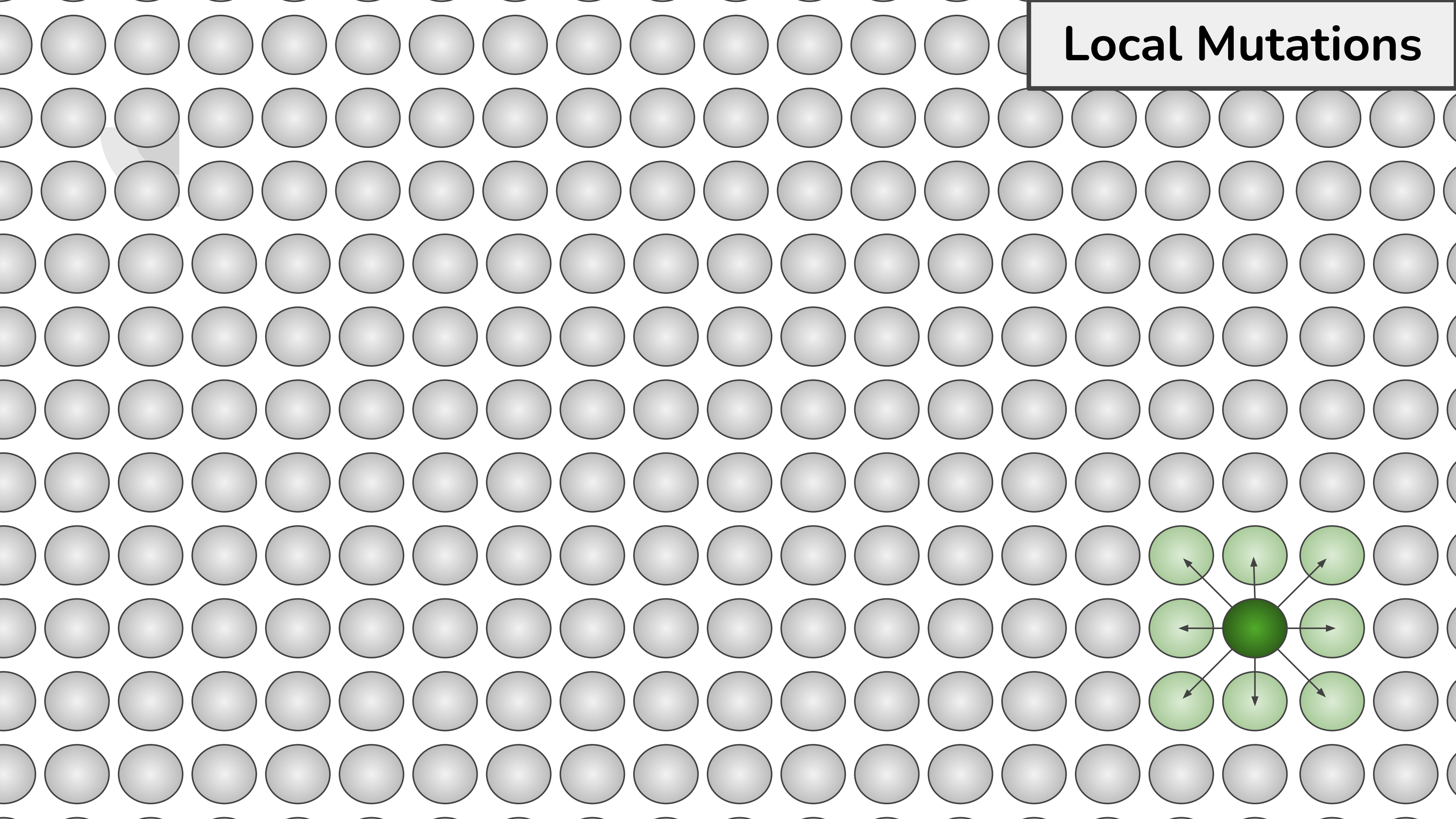


Random Inputs

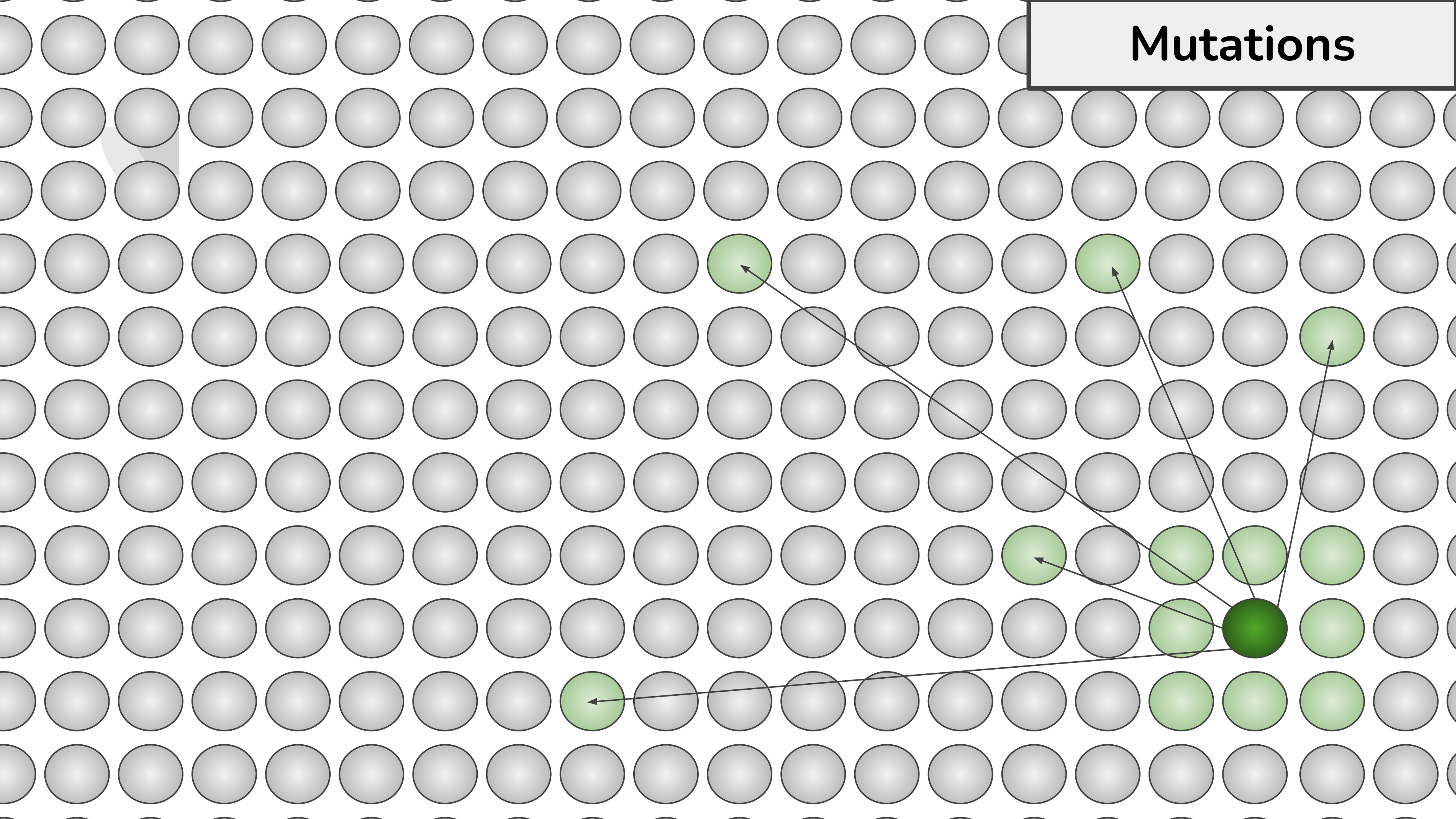


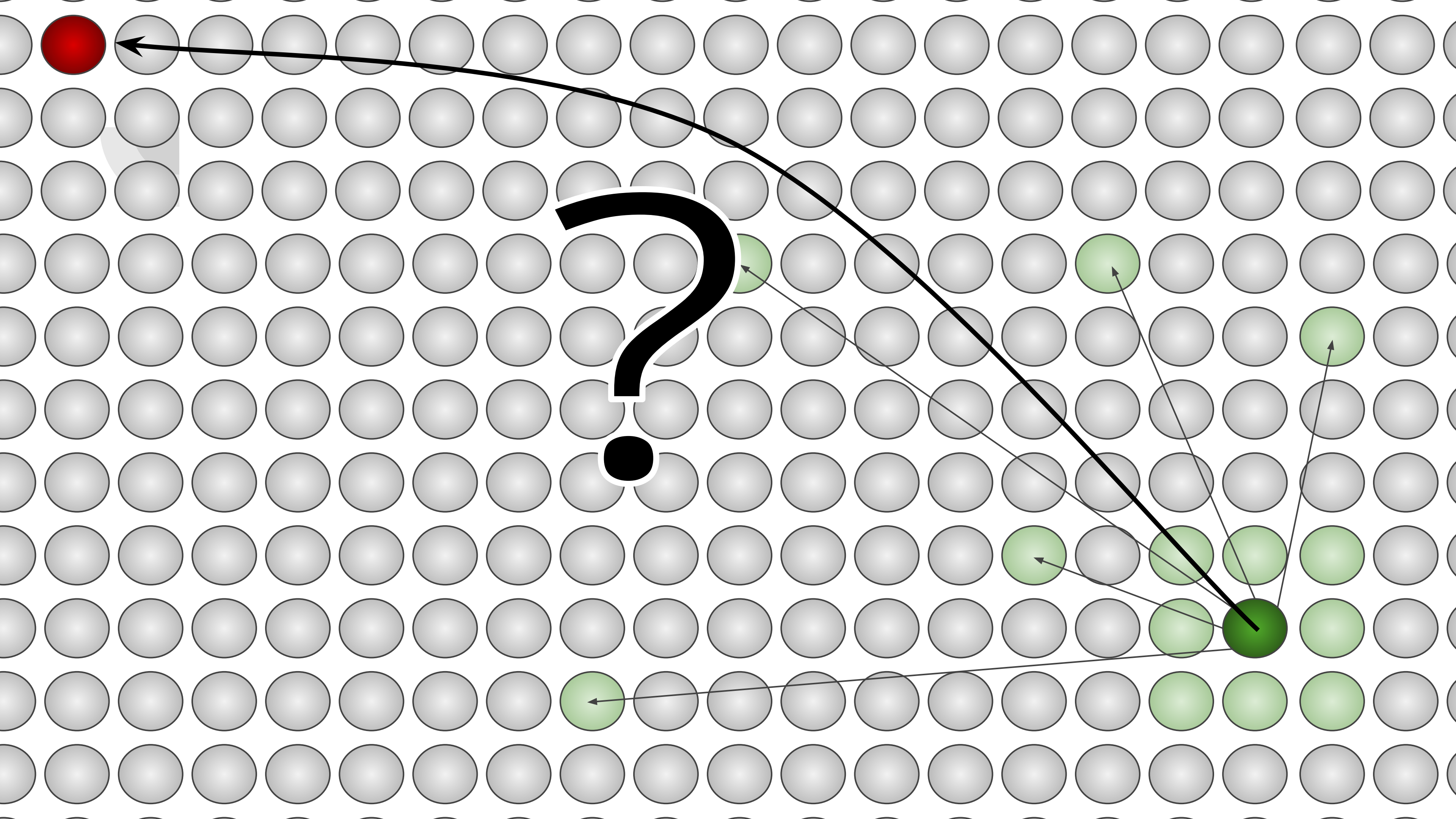
Input Seed

Local Mutations

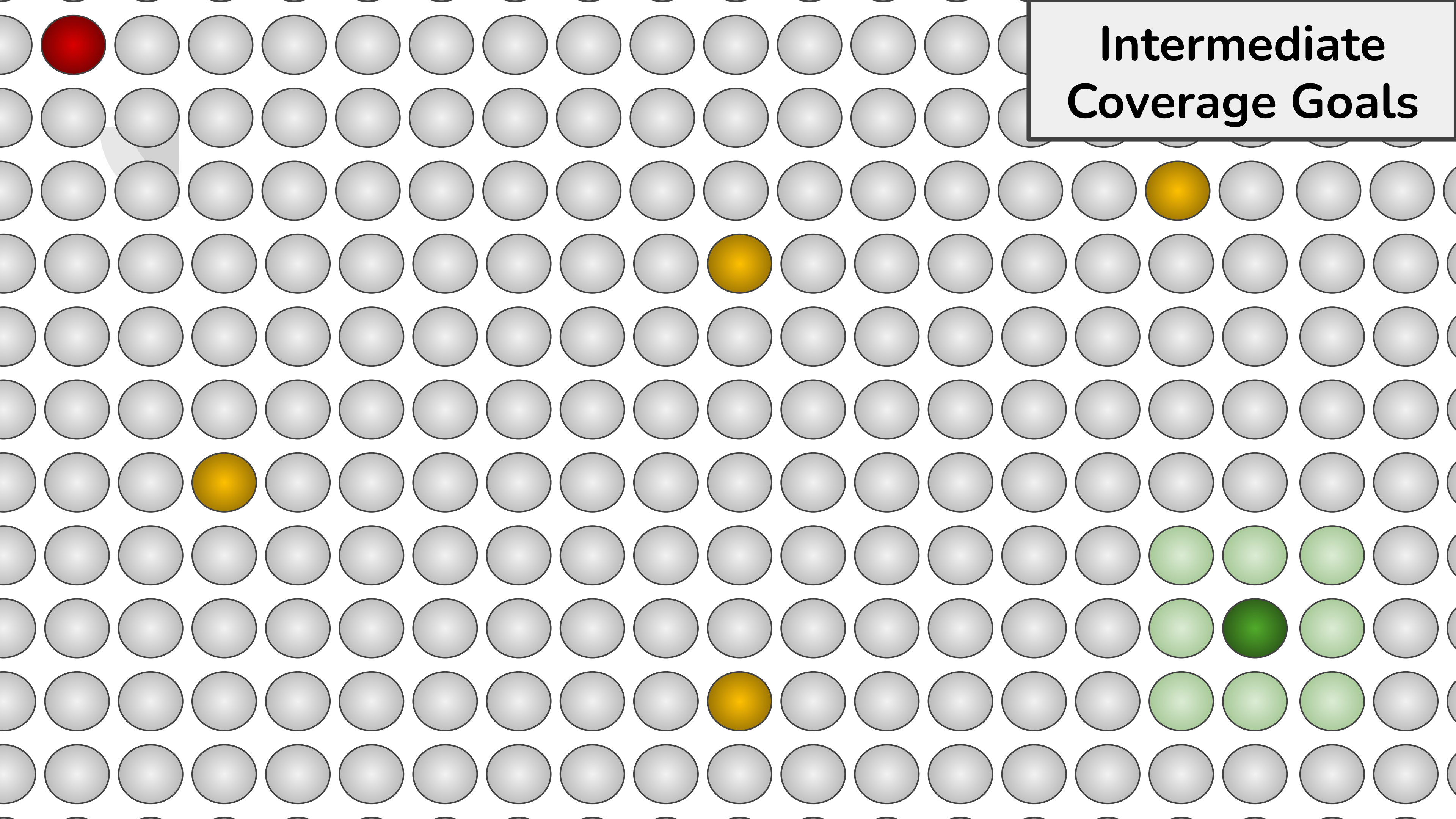


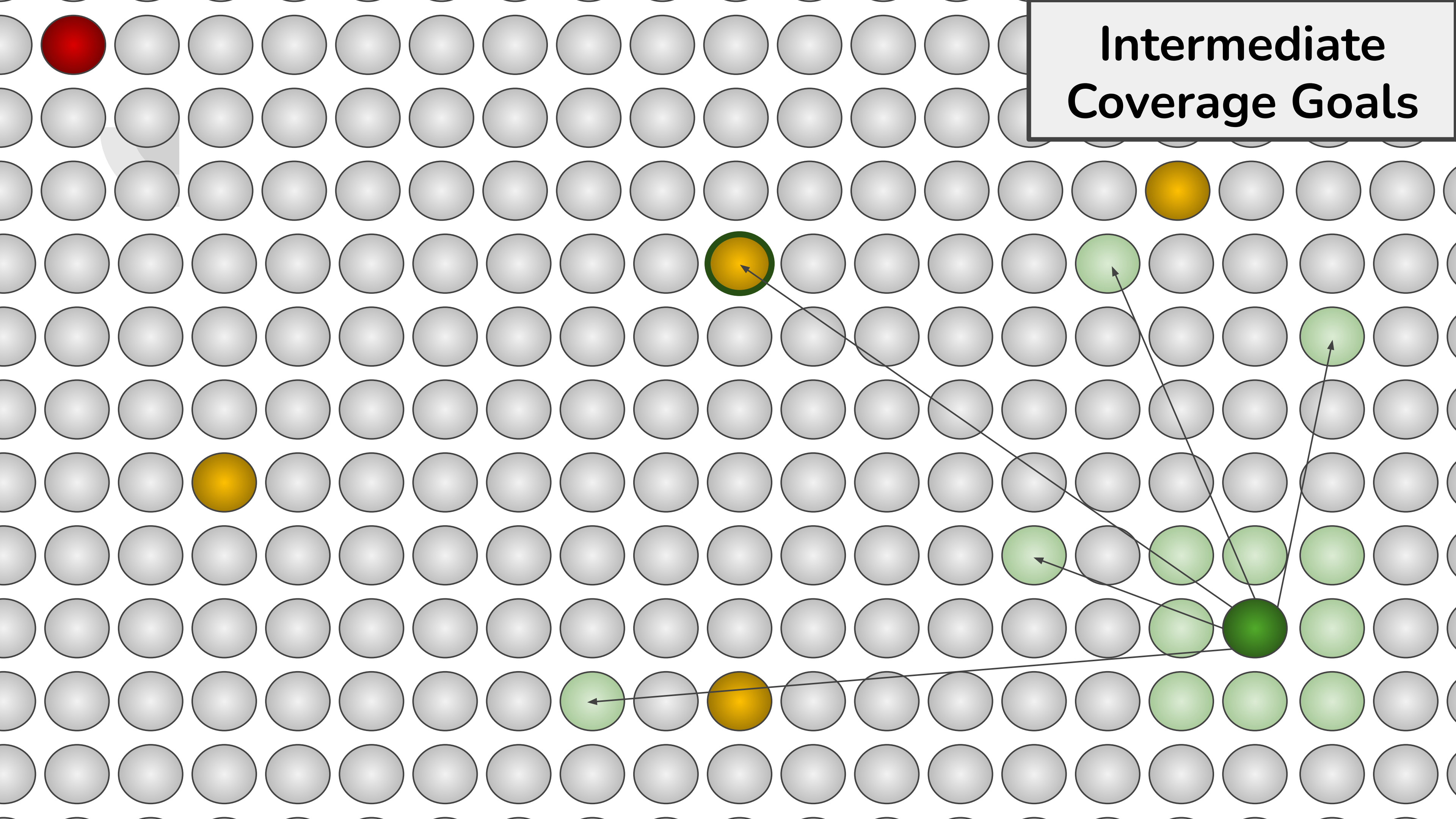
Mutations



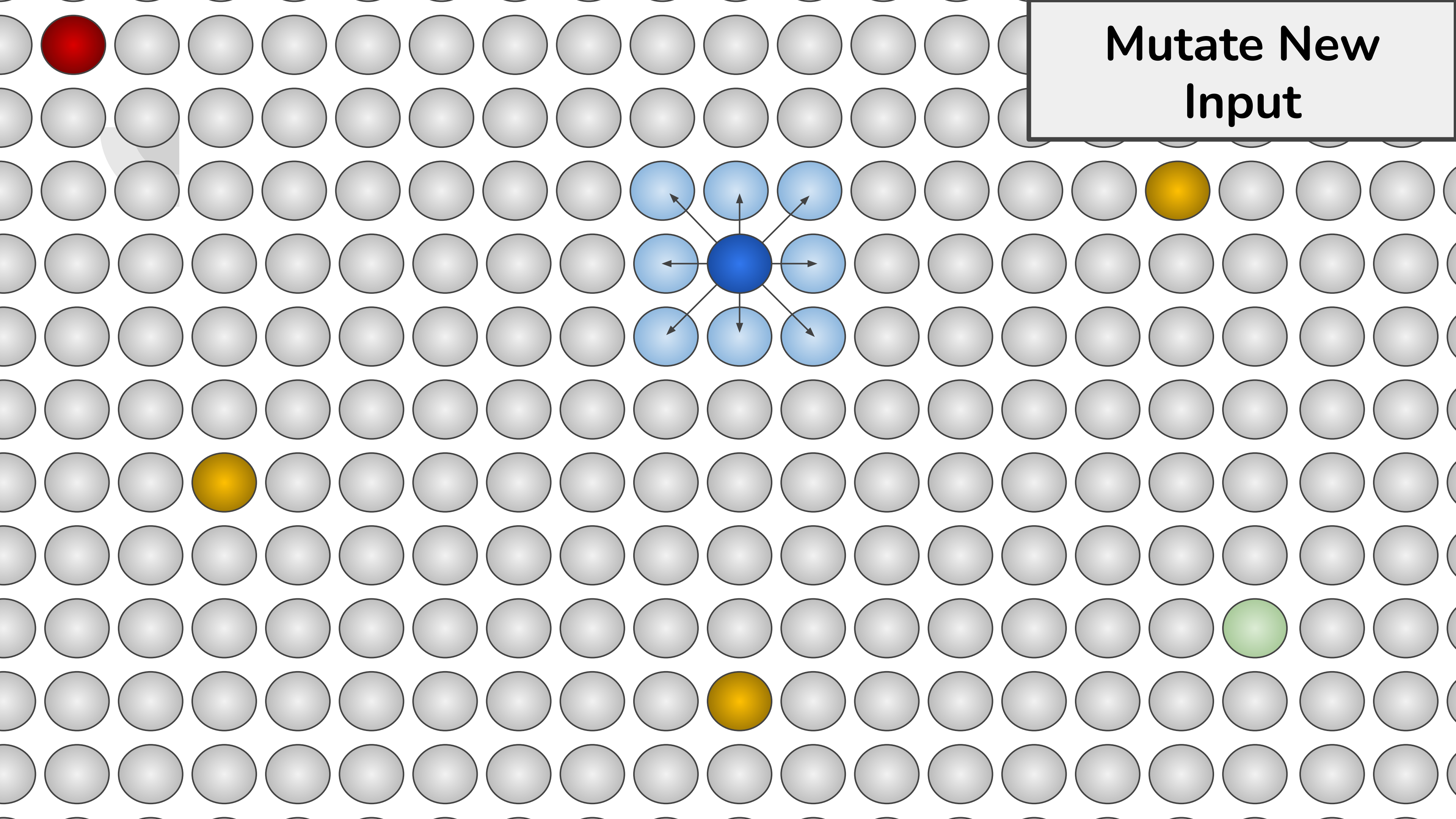


Intermediate Coverage Goals

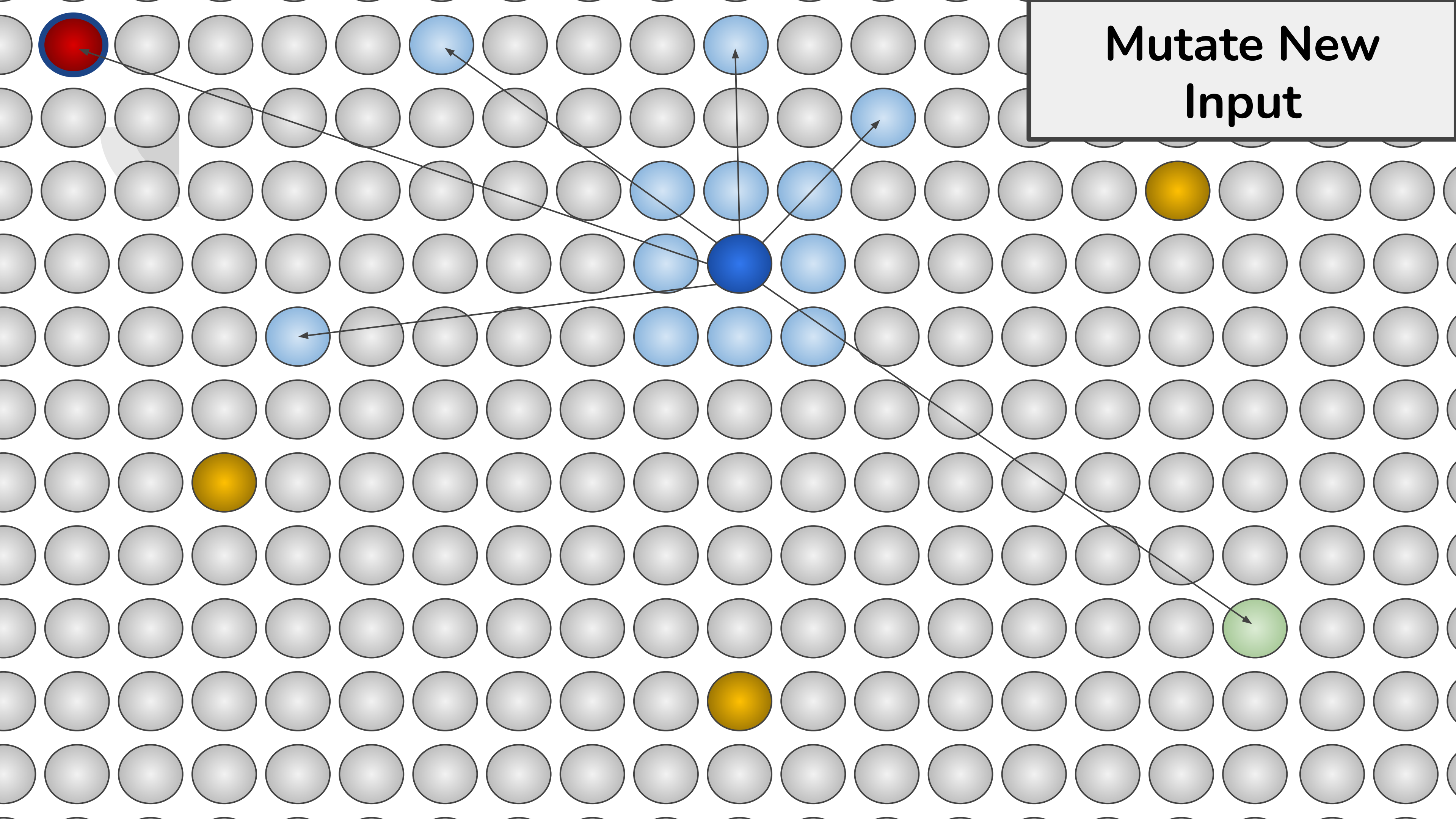




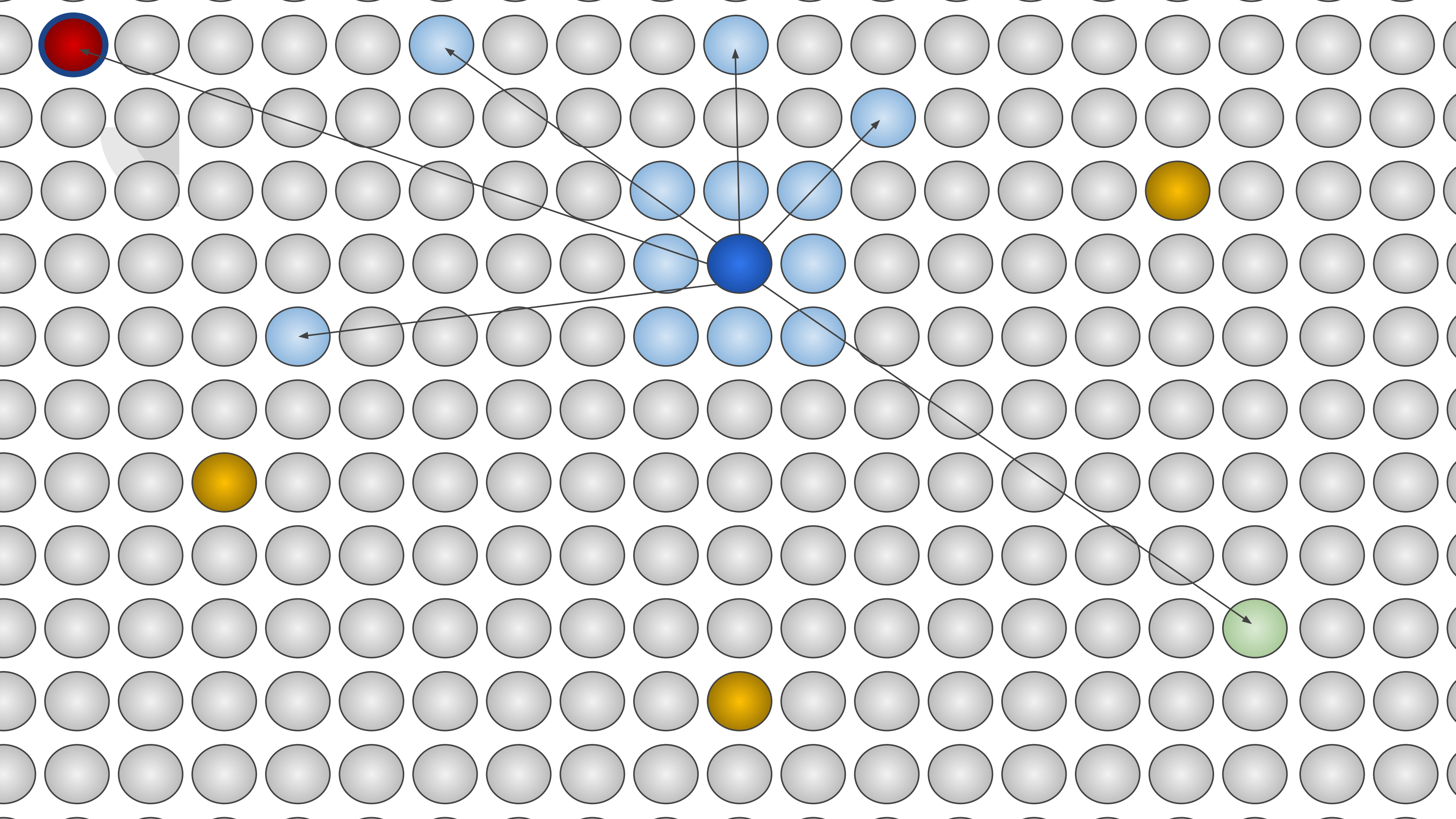
Intermediate Coverage Goals



Mutate New Input

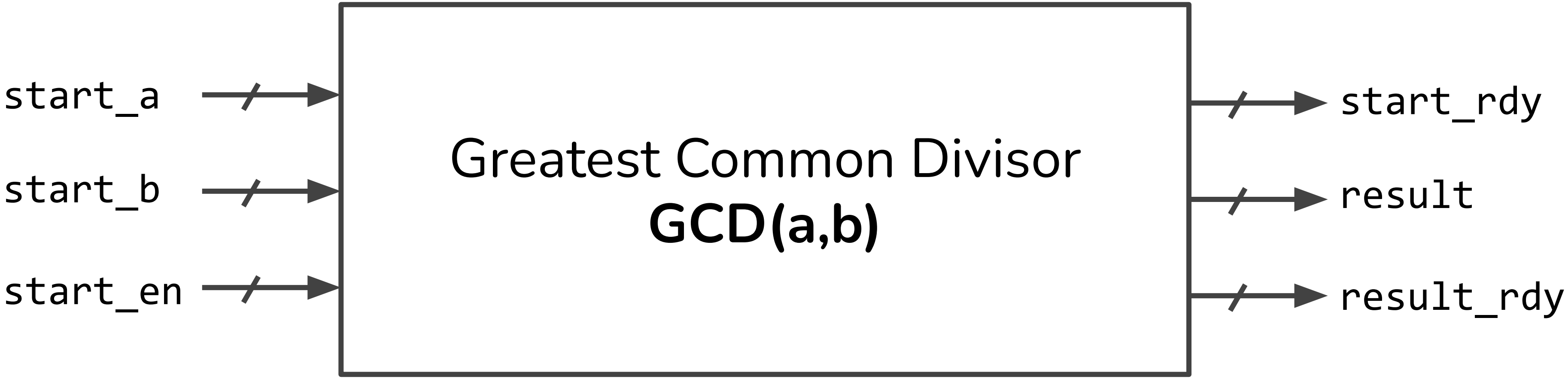


Mutate New Input

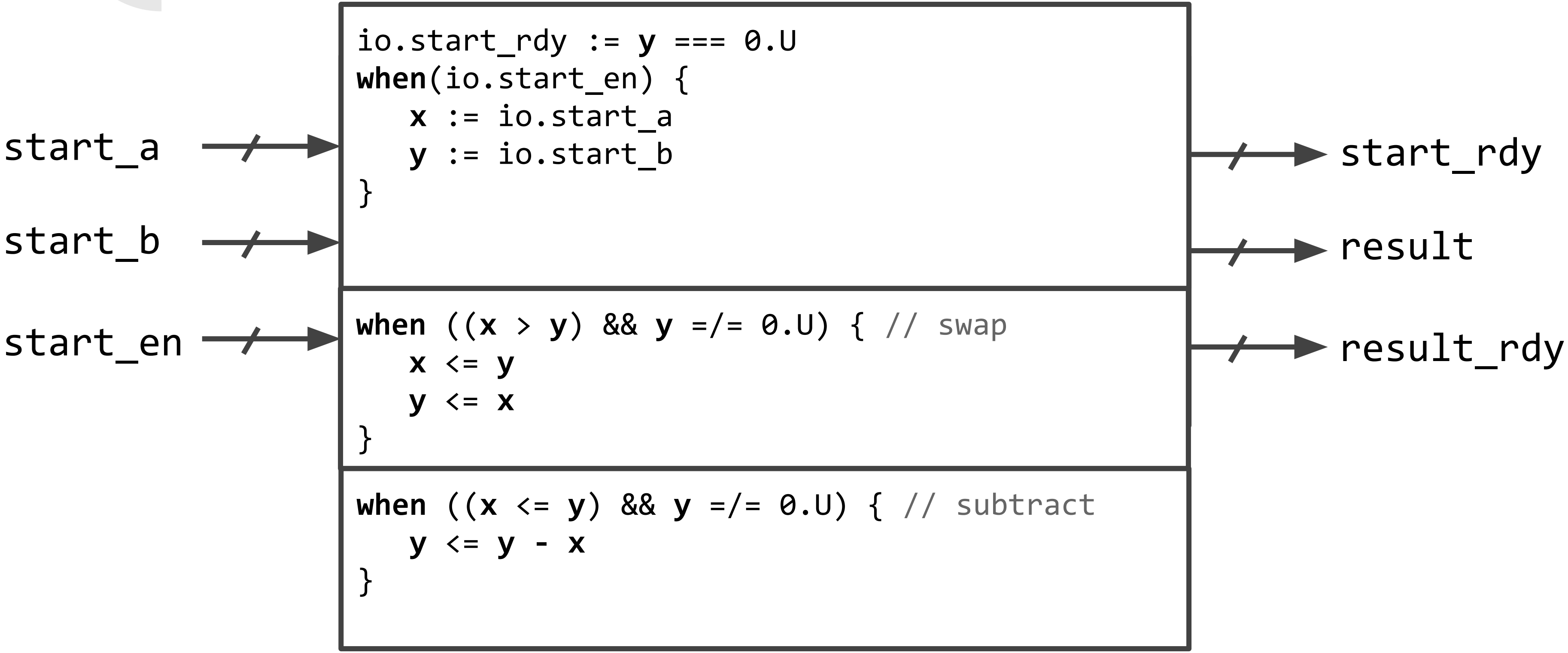


Coverage-Directed Fuzzing: An Example

Fuzzing Example: GCD



Fuzzing Example: GCD

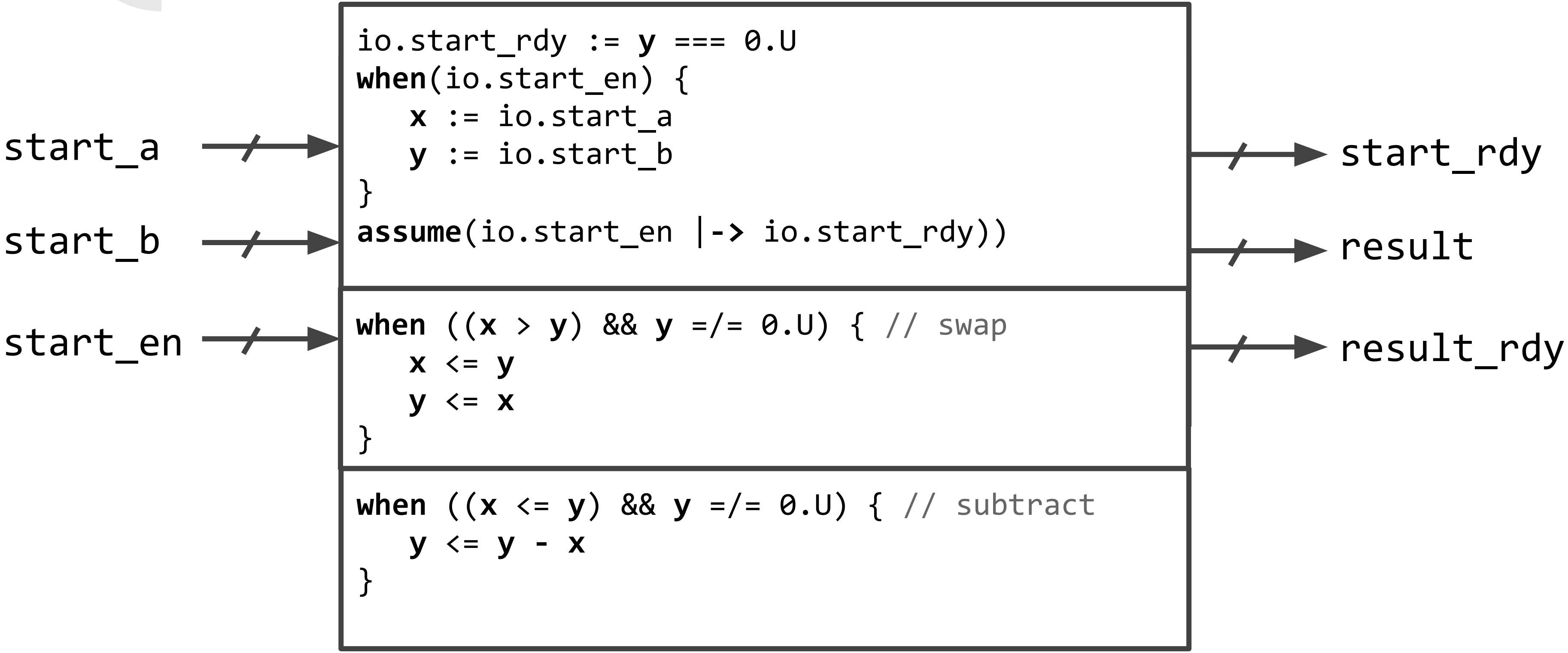


```
io.start_rdy := y === 0.U
when(io.start_en) {
   x := io.start_a
   y := io.start_b
}
```

```
when ((x > y) && y =/= 0.U) { // swap
   x <= y
   y <= x
}
```

```
when ((x <= y) && y =/= 0.U) { // subtract
   y <= y - x
}
```

Fuzzing Example: GCD

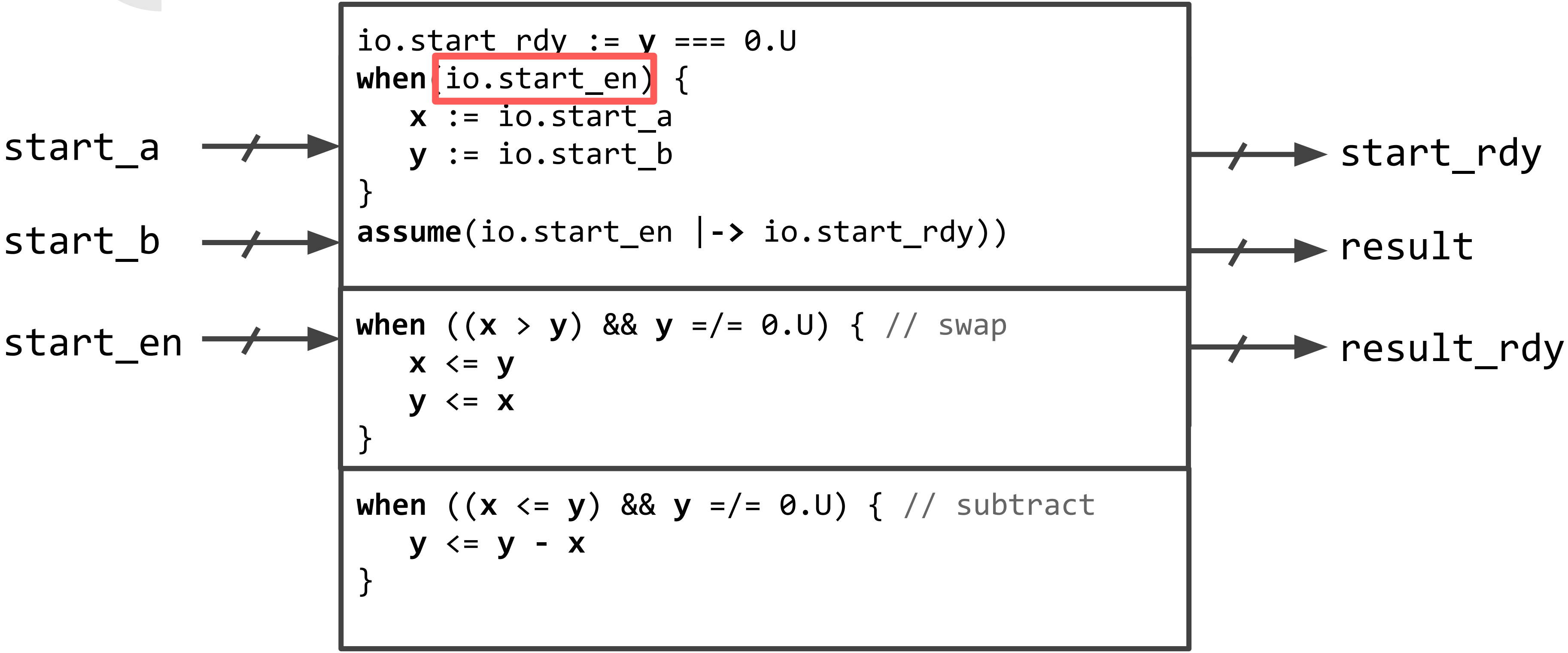


```
io.start_rdy := y === 0.U
when(io.start_en) {
   x := io.start_a
   y := io.start_b
}
assume(io.start_en |-> io.start_rdy))
```

```
when ((x > y) && y =/= 0.U) { // swap
   x <= y
   y <= x
}
```

```
when ((x <= y) && y =/= 0.U) { // subtract
   y <= y - x
}
```

Fuzzing Example: GCD



start_a

start_b

start_en

```
io.start_rdy := y === 0.U
when(io.start_en) {
   x := io.start_a
   y := io.start_b
}
assume(io.start_en |-> io.start_rdy))
```

```
when ((x > y) && y =/= 0.U) { // swap
   x <= y
   y <= x
}
```

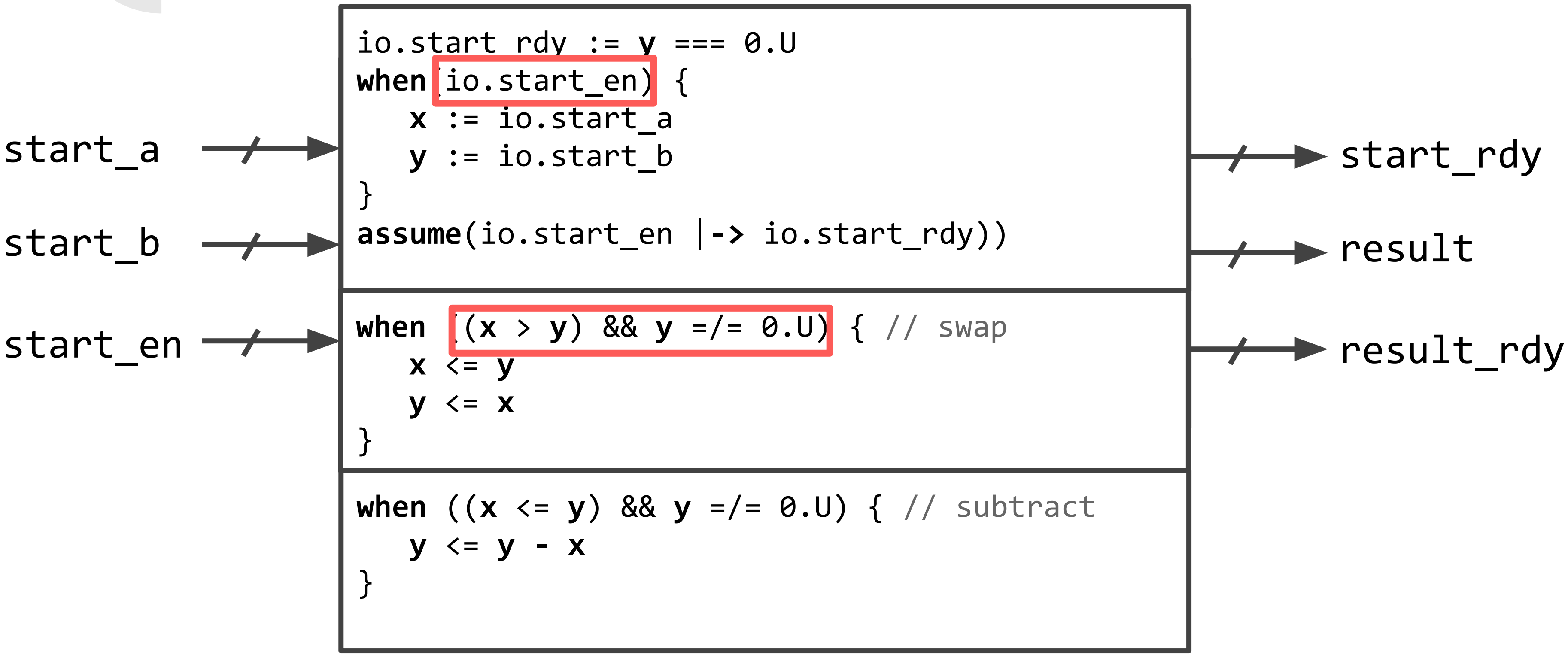
```
when ((x <= y) && y =/= 0.U) { // subtract
   y <= y - x
}
```

start_rdy

result

result_rdy

Fuzzing Example: GCD

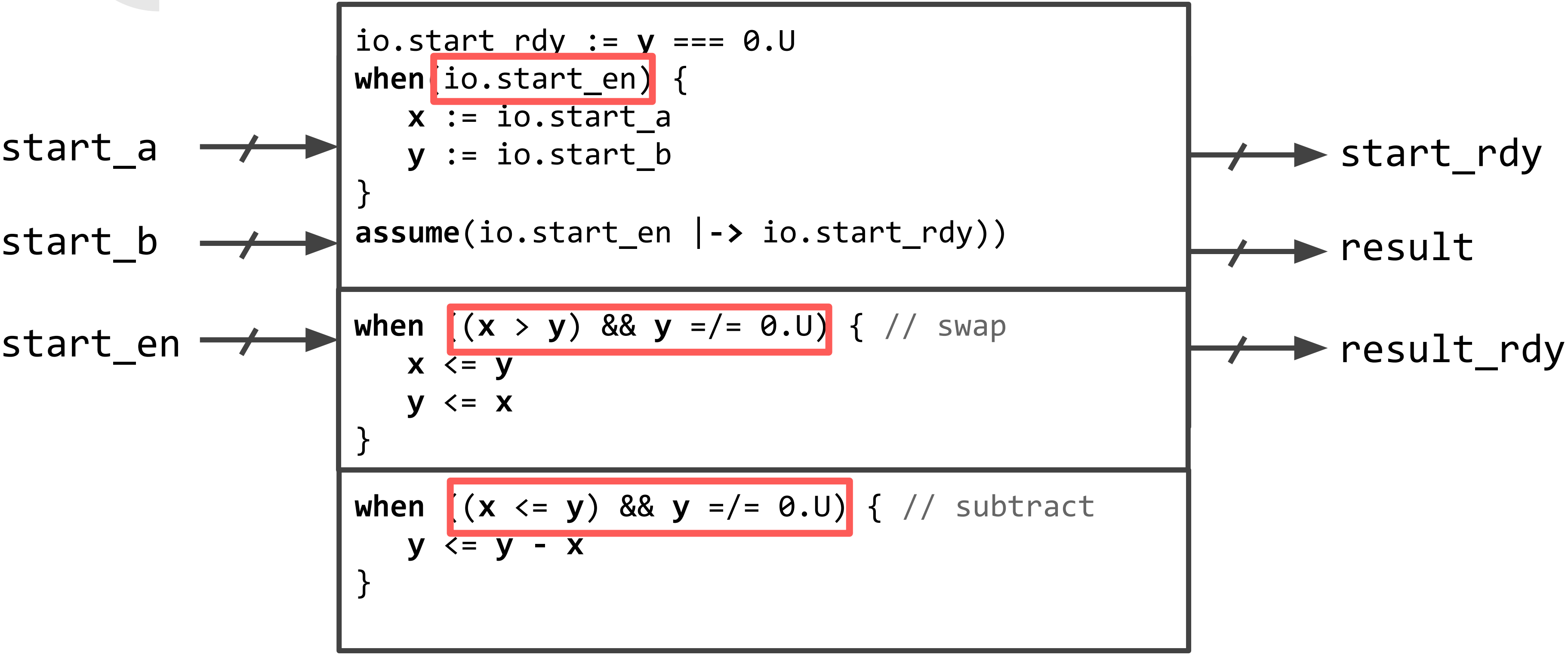


```
io.start_rdy := y === 0.U
when(io.start_en) {
   x := io.start_a
   y := io.start_b
}
assume(io.start_en |-> io.start_rdy))
```

```
when ((x > y) && y =/= 0.U) { // swap
   x <= y
   y <= x
}
```

```
when ((x <= y) && y =/= 0.U) { // subtract
   y <= y - x
}
```

Fuzzing Example: GCD



start_a

start_b

start_en

```
io.start_rdy := y === 0.U
when (io.start_en) {
   x := io.start_a
   y := io.start_b
}
assume(io.start_en |-> io.start_rdy))
```

```
when ((x > y) && y =/= 0.U) { // swap
   x <= y
   y <= x
}
```

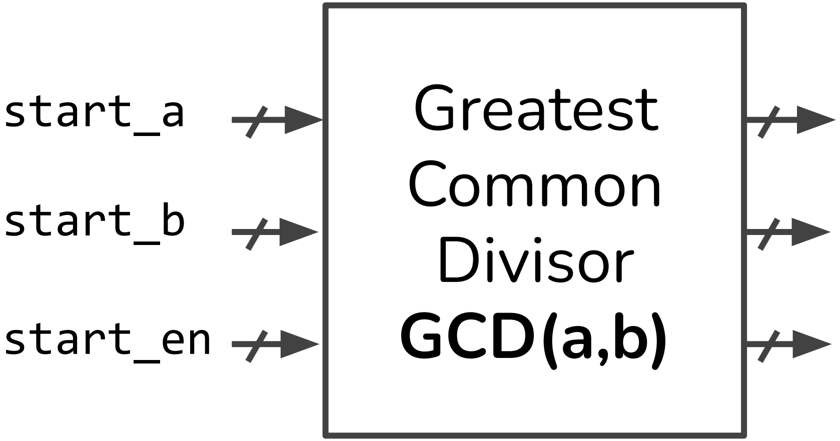
```
when ((x <= y) && y =/= 0.U) { // subtract
   y <= y - x
}
```

start_rdy

result

result_rdy

Fuzzing Example: GCD



io.start_en	-
(**x** > **y**) && **y** =/= 0.U	-
(**x** <= **y**) && **y** =/= 0.U	-

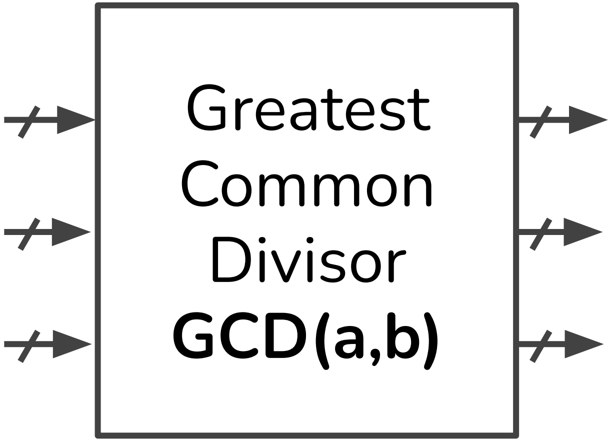
Fuzzing Example: GCD

Input 0

00000000000000000000000000000000 start_a

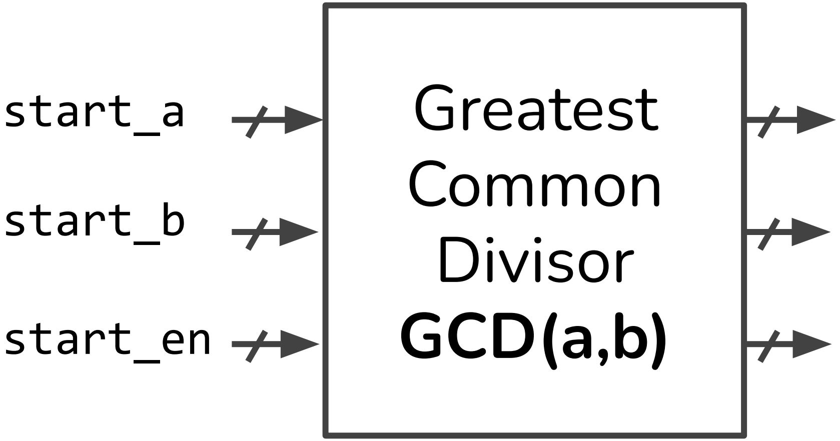
00000000000000000000000000000000 start_b

0 start_en



io.start_en	-
(**x** > **y**) && **y** =/= 0.U	-
(**x** <= **y**) && **y** =/= 0.U	-

Fuzzing Example: GCD



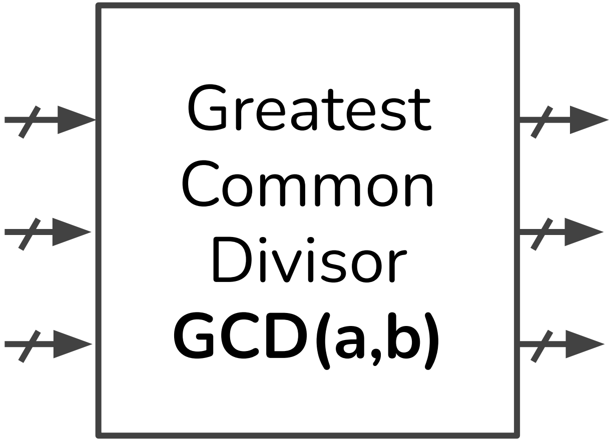
```
00000000000000000000000000000000 start_a
00000000000000000000000000000000 start_b
                               0 start_en
```

`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

+ 4 more cycles of all zeros!

Fuzzing Example: GCD

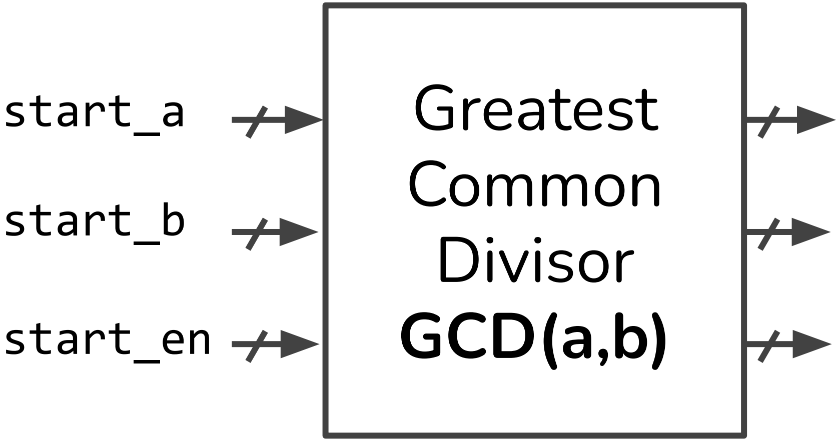
```
10000000000000000000000000000000 start_a
00000000000000000000000000000000 start_b
                               0 start_en
```



`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

```
01000000000000000000000000000000
00000000000000000000000000000000
                               0
```



start_a

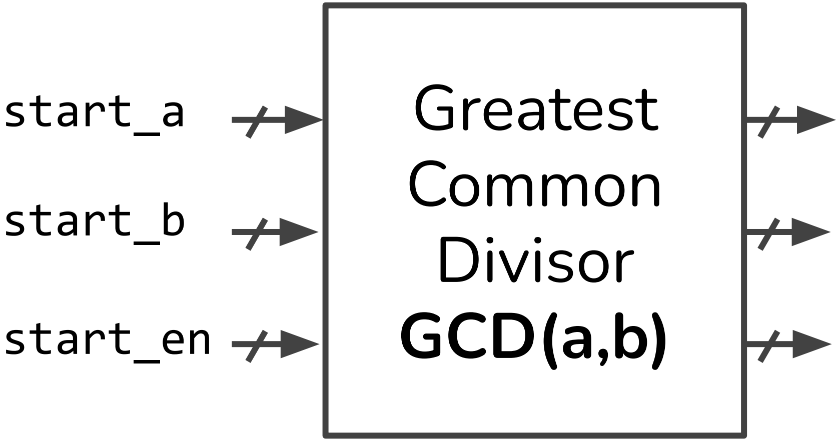
start_b

start_en

Greatest Common Divisor **GCD(a,b)**

`io.start_en`	-
`(`**`x`**` > `**`y`**`)  && `**`y`**` =/= 0.U`	-
`(`**`x`**` <= `**`y`**`) && `**`y`**` =/= 0.U`	-

Fuzzing Example: GCD

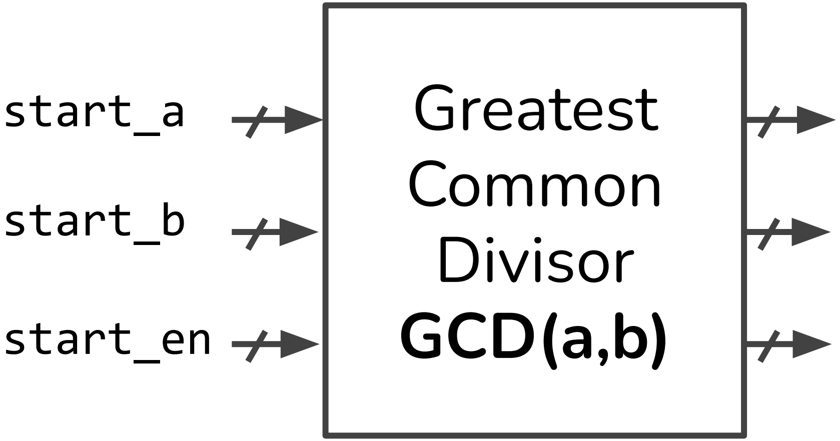


00100000000000000000000000000000 start_a
00000000000000000000000000000000 start_b
0 start_en
Greatest Common Divisor **GCD(a,b)**

`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

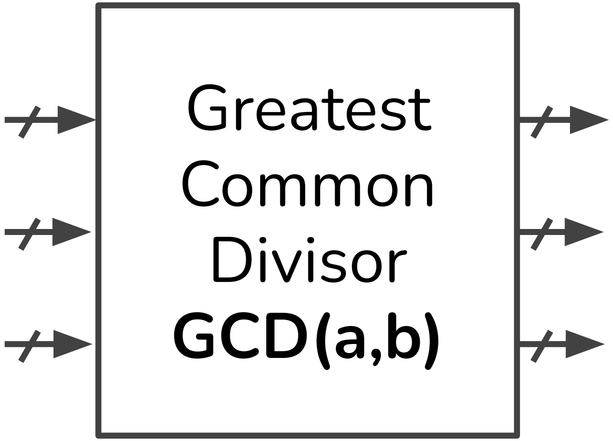
```
00010000000000000000000000000000
00000000000000000000000000000000
                               0
```



`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

```
00000000000000000000000000000001 start_a
00000000000000000000000000000000 start_b
                               0 start_en
```

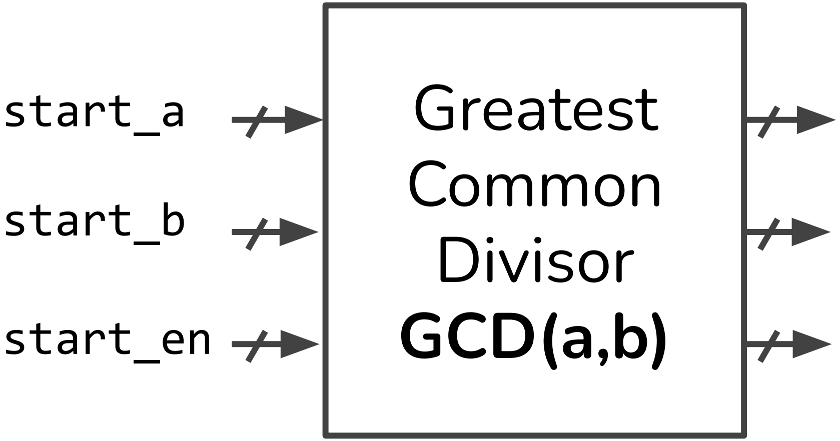


Greatest Common Divisor **GCD(a,b)**

`io.start_en`	-
`(`**`x`**` > `**`y`**`)  && `**`y`**` =/= 0.U`	-
`(`**`x`**` <= `**`y`**`) && `**`y`**` =/= 0.U`	-

Fuzzing Example: GCD

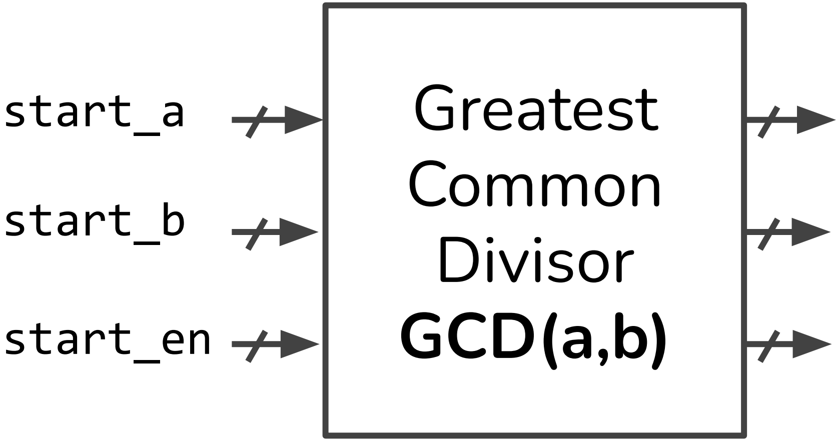
```
00000000000000000000000000000000
10000000000000000000000000000000
0
```



`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

```
00000000000000000000000000000000
01000000000000000000000000000000
                               0
```



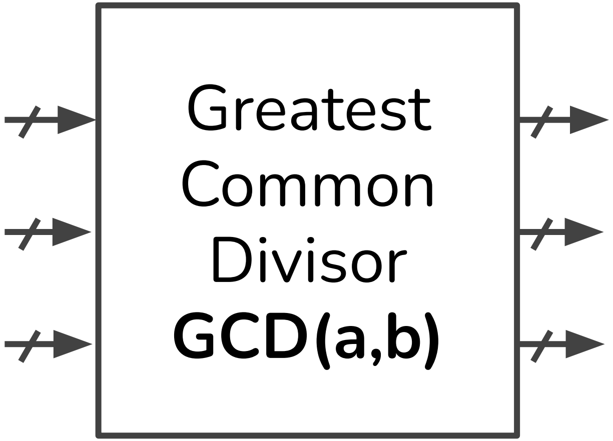
`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

`00000000000000000000000000000000` start_a

`00100000000000000000000000000000` start_b

`0` start_en



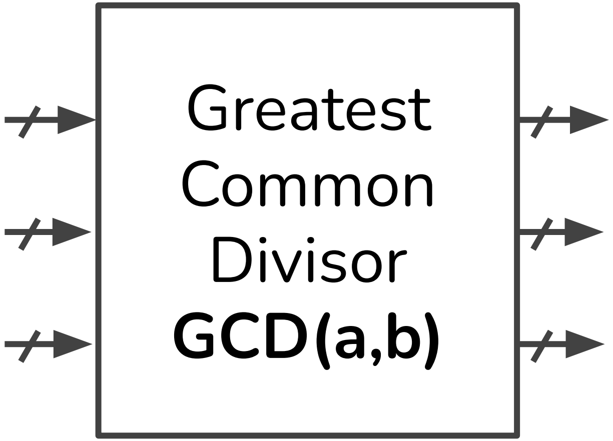
`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

`00000000000000000000000000000000` start_a

`00000000000000000000000000000001` start_b

`0` start_en



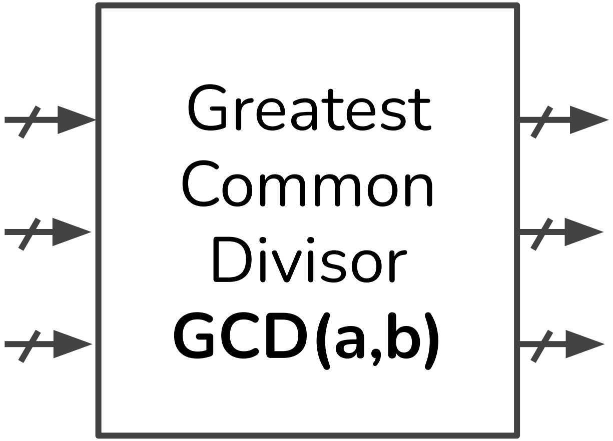
io.start_en	-
(**x** > **y**) && **y** =/= 0.U	-
(**x** <= **y**) && **y** =/= 0.U	-

Fuzzing Example: GCD

00000000000000000000000000000000 start_a

00000000000000000000000000000000 start_b

1 start_en



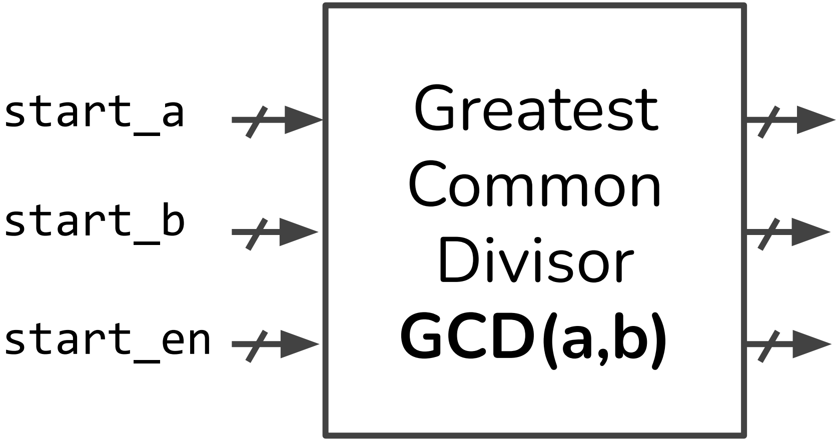
`io.start_en`	-
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

00000000000000000000000000000000

00000000000000000000000000000000

1



start_a

start_b

start_en

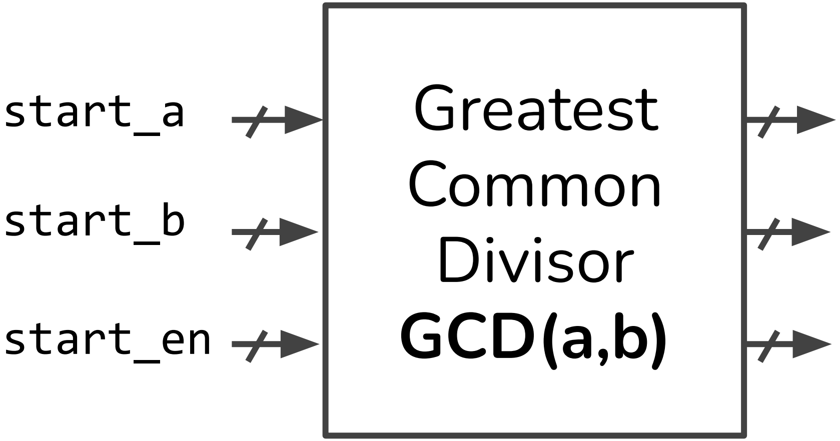
Greatest Common Divisor **GCD(a,b)**

`io.start_en`	✅
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

Input 1

```
00000000000000000000000000000000
00000000000000000000000000000000
                               1
```

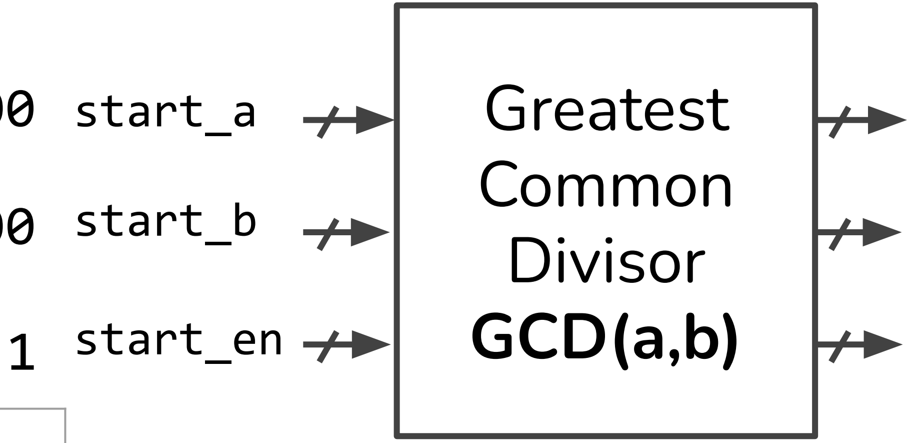


`io.start_en`	✅
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Fuzzing Example: GCD

Input 1

```
00000000000000000000000000000000

00000000000000000000000000000000
```



`io.start_en`	✅
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	-

Generated by flipping a single bit in Input 0

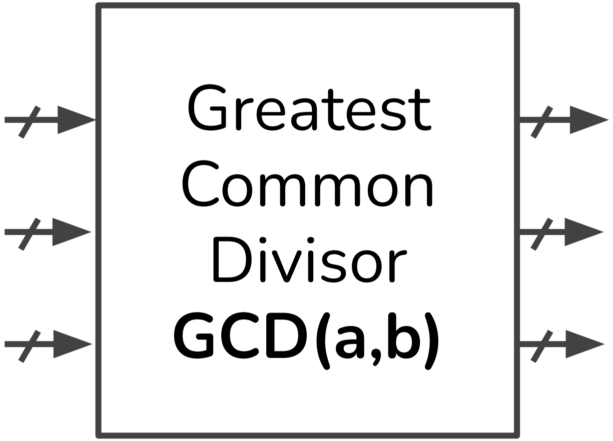
Fuzzing Example: GCD

Input 2

`00000000000000000000000000000000` start_a

`00000000000000000000000011111111` start_b

`1` start_en



Greatest Common Divisor **GCD(a,b)**

`io.start_en`	✅
`(x > y)  && y =/= 0.U`	-
`(x <= y) && y =/= 0.U`	✅

Generated by flipping 16 bit on byte offsets in Input 0

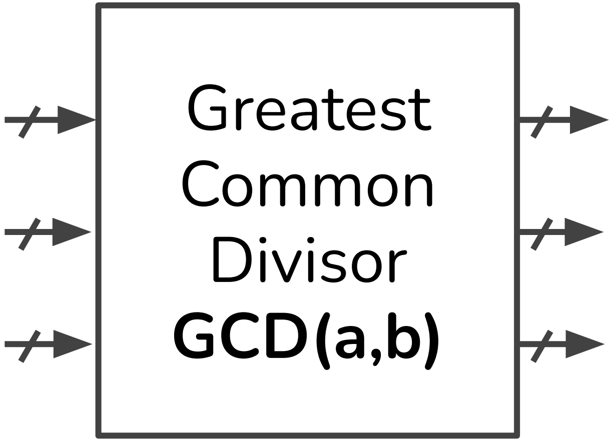
Fuzzing Example: GCD

Input 3

`10000000000000000000000000000000` `start_a`

`00000000000000000000000011111111` `start_b`

`1` `start_en`

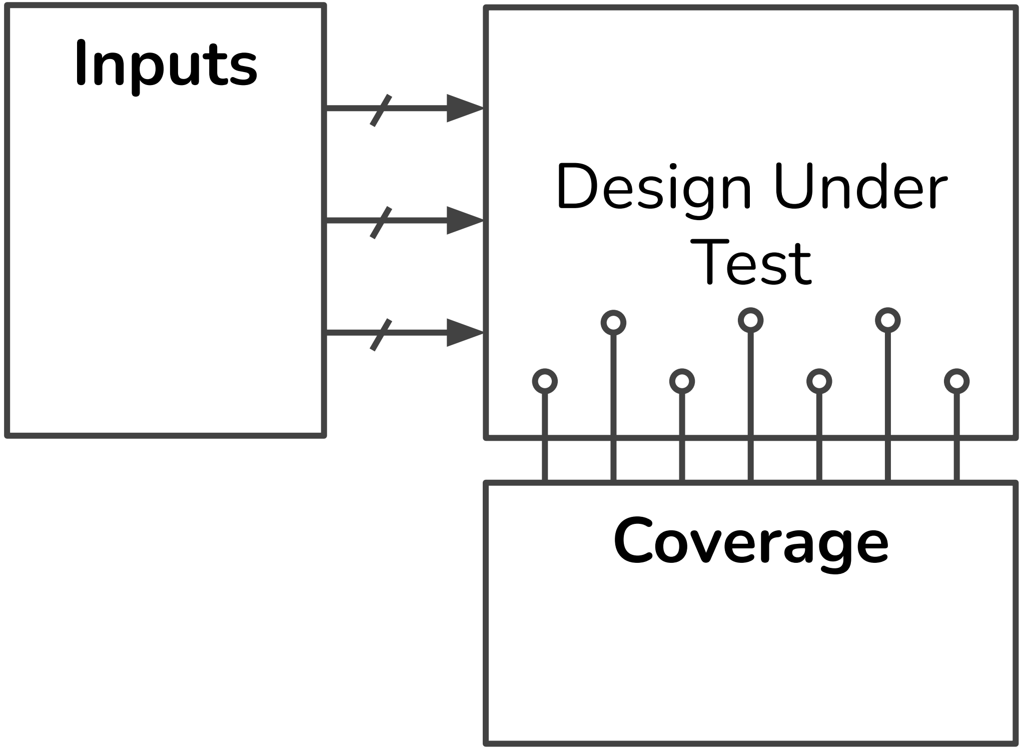


`io.start_en`	✅
`(x > y)  && y =/= 0.U`	✅
`(x <= y) && y =/= 0.U`	✅

Generated by flipping a single bit in Input 2

Implementation

Deterministic Test Execution

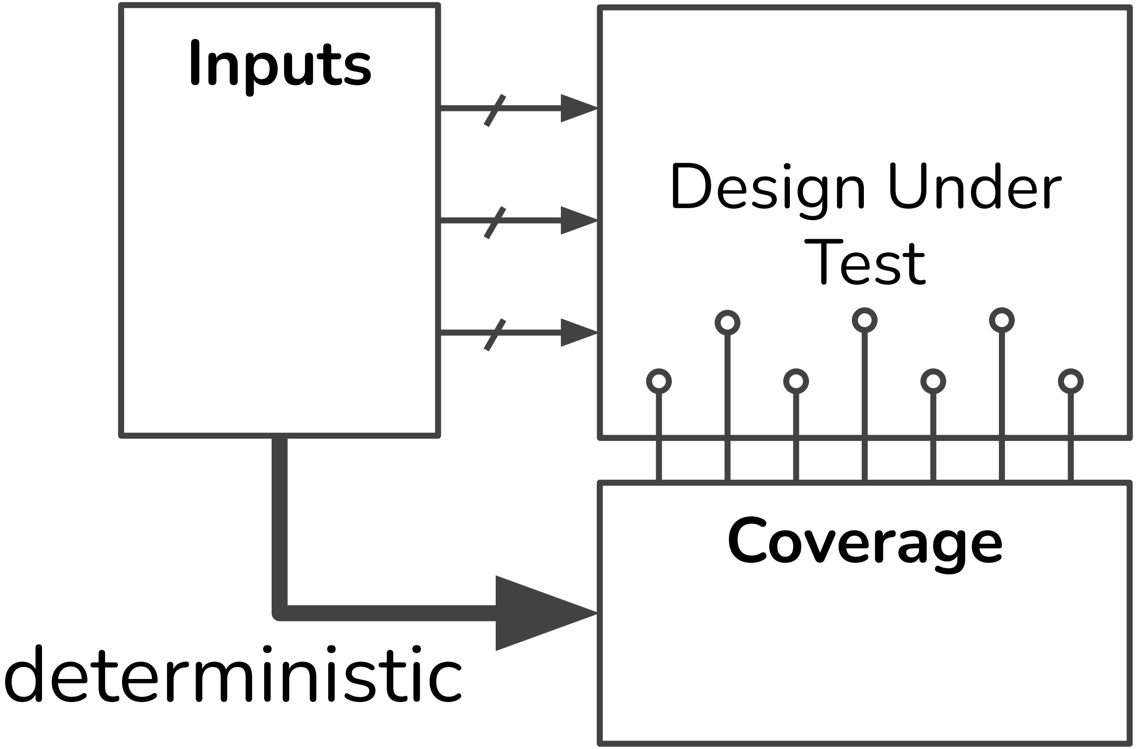


Inputs

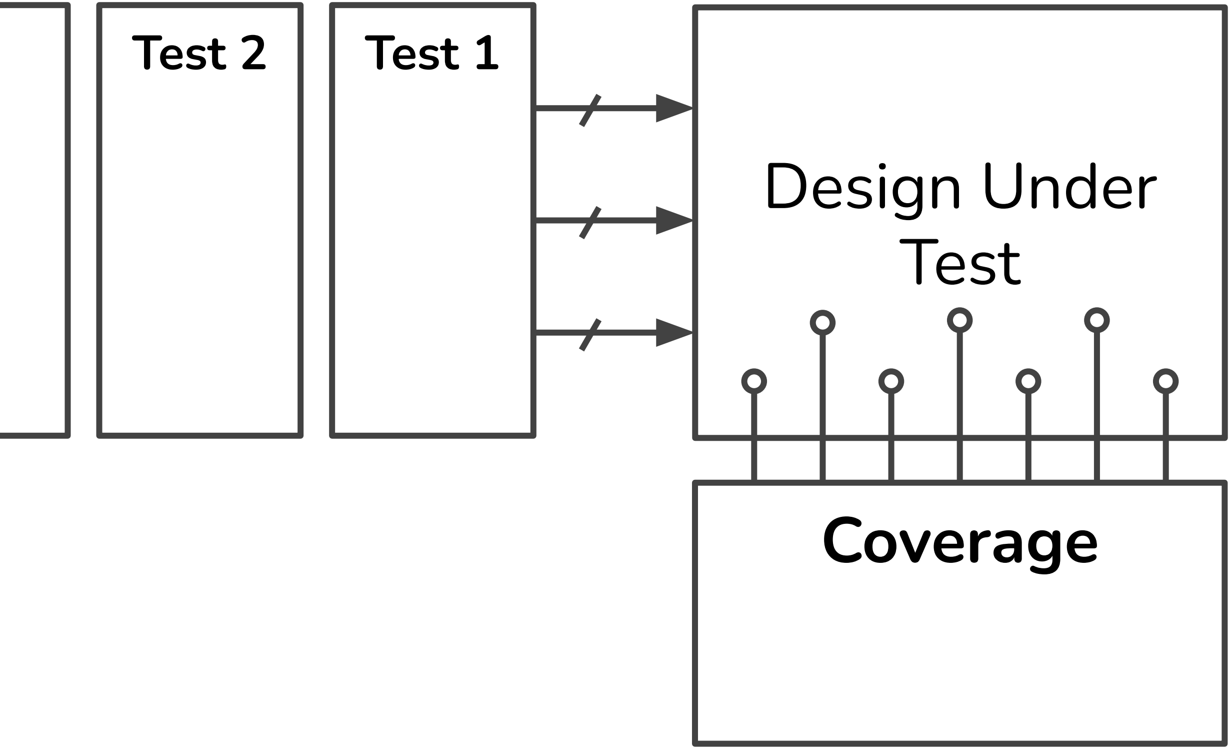
Design Under Test

Coverage

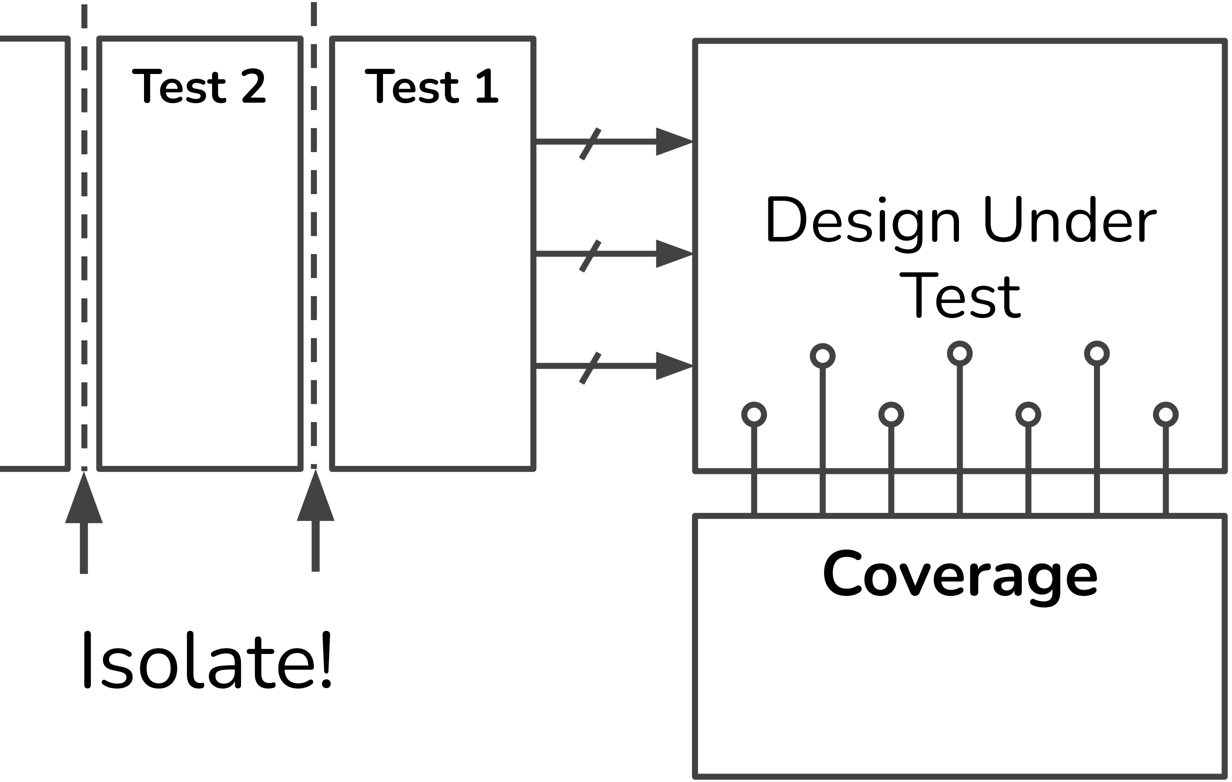
Deterministic Test Execution



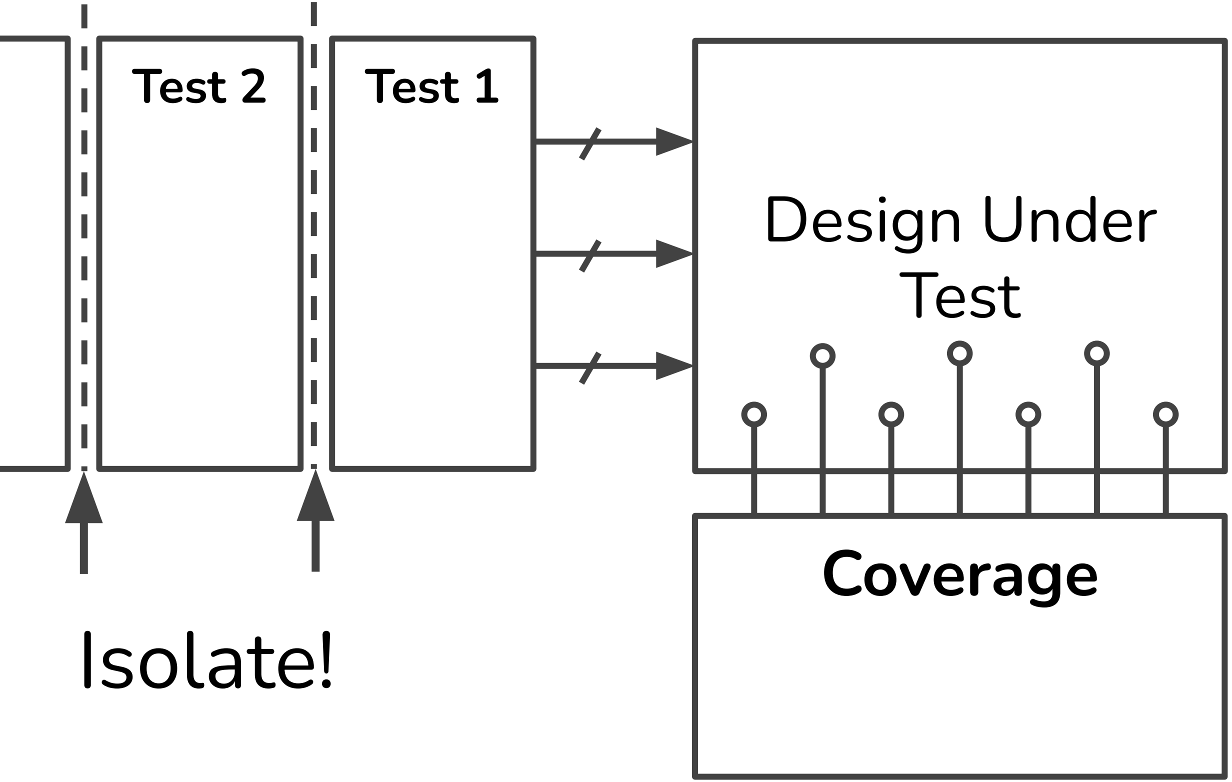
Deterministic Test Execution



Deterministic Test Execution



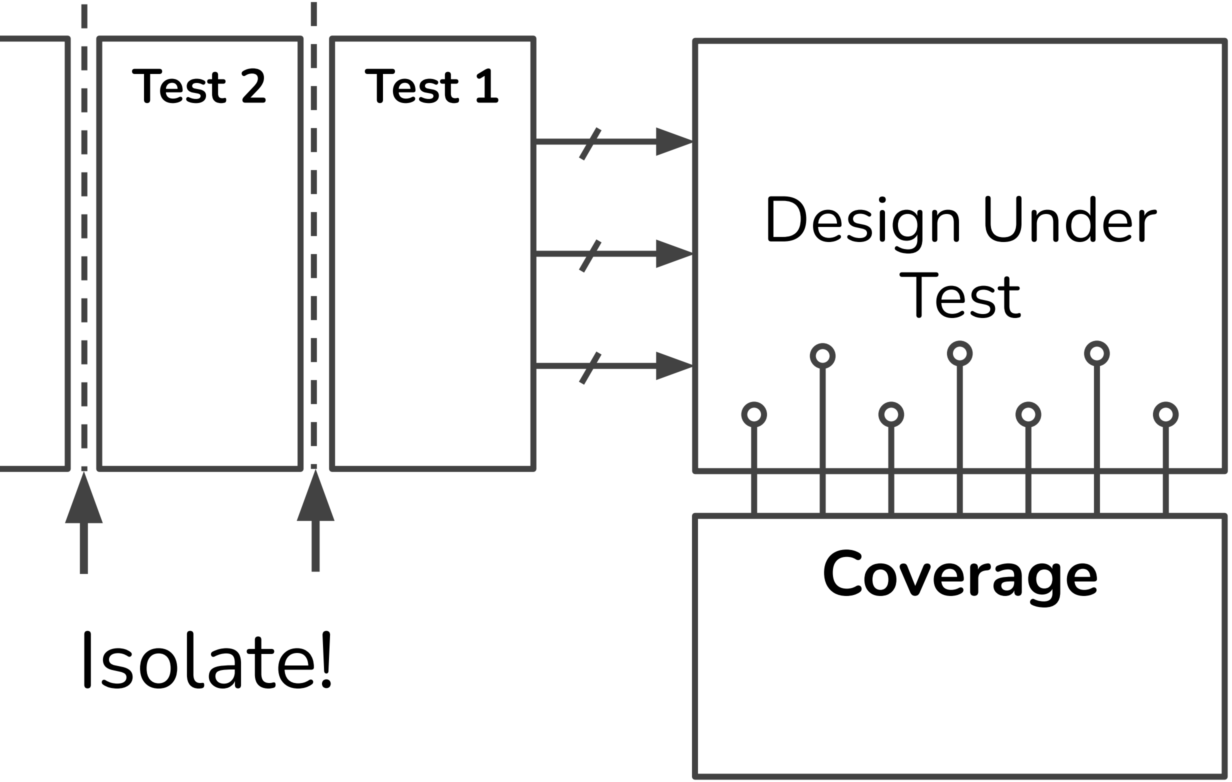
Deterministic Test Execution



Isolate!

Two Problems:

Deterministic Test Execution

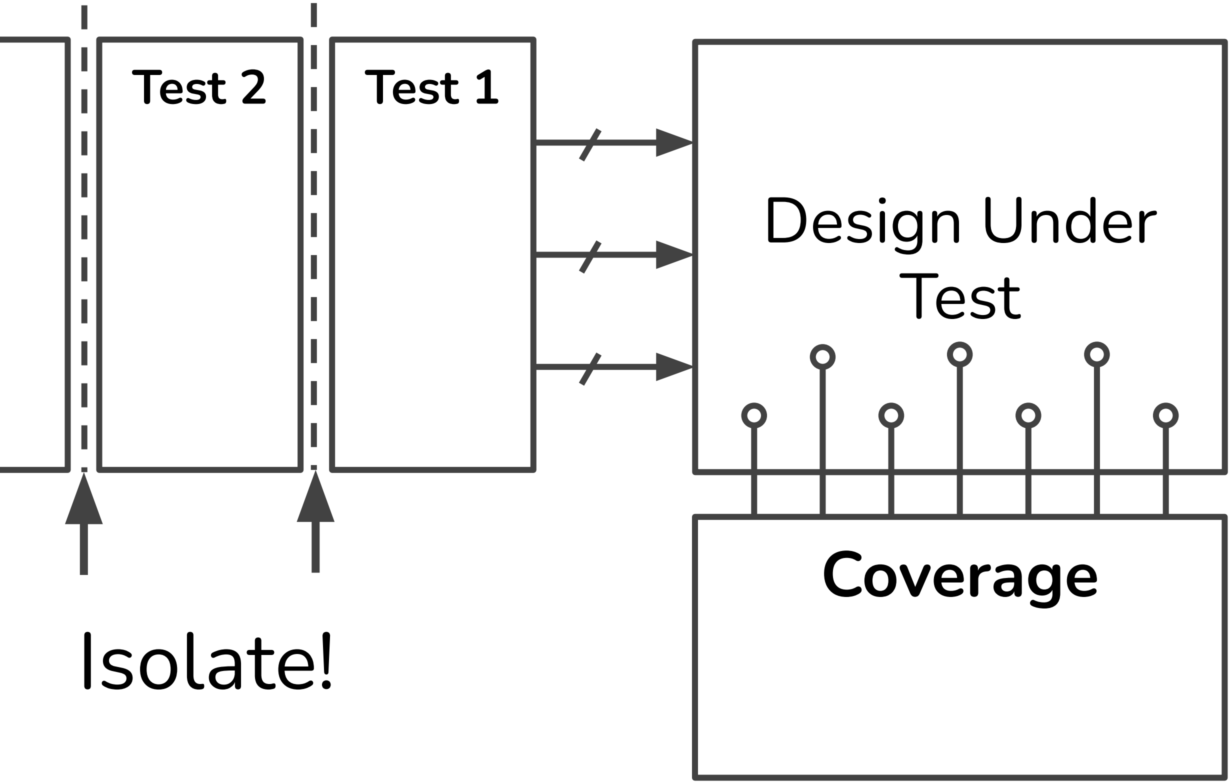


Isolate!

Two Problems:

registers with undefined reset (XXXX)

Deterministic Test Execution



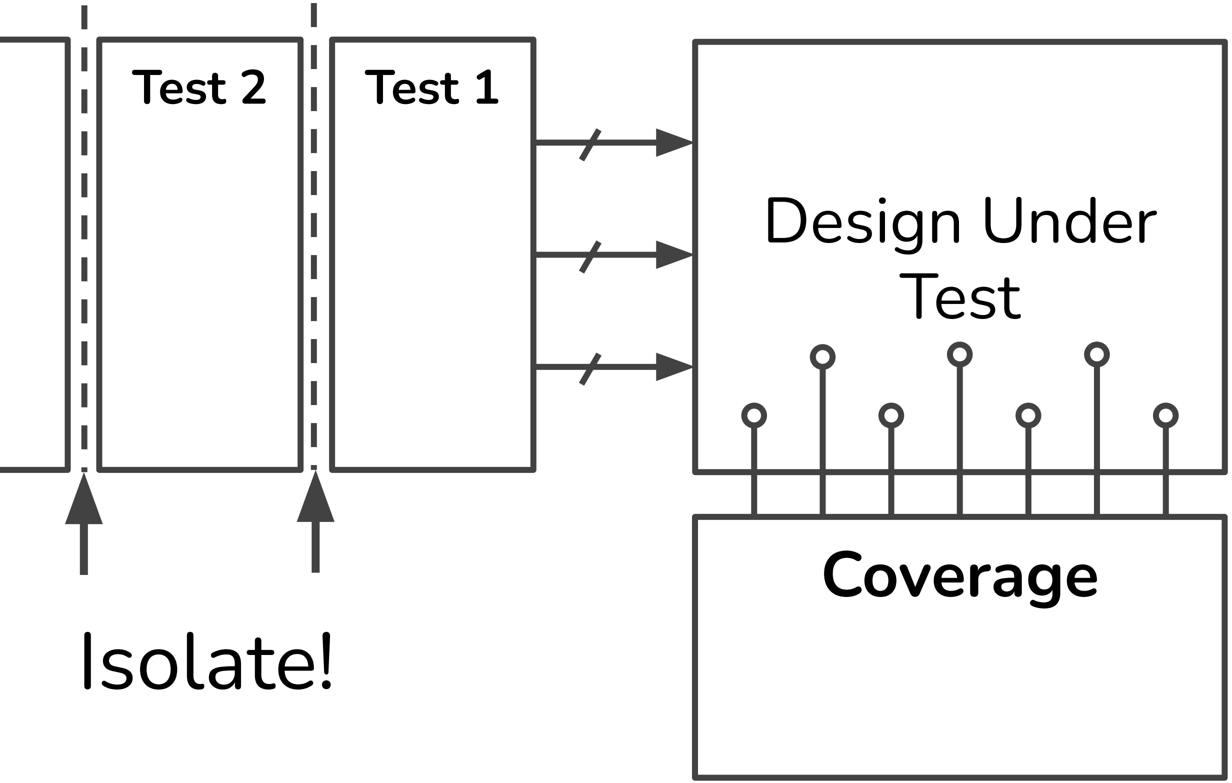
Isolate!

Two Problems:

registers with undefined reset (XXXX)

memories

Deterministic Test Execution



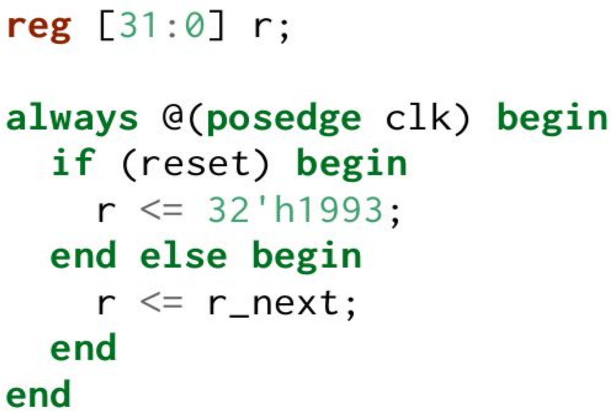
Isolate!

Two Problems:

registers with undefined reset (XXXX)

memories

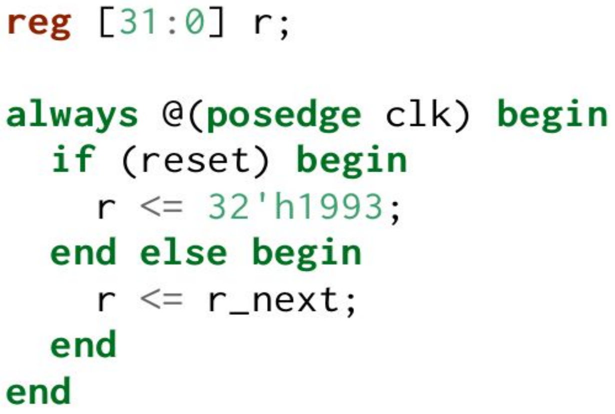
Meta Reset



```
reg [31:0] r;

always @(posedge clk) begin
  if (reset) begin
    r <= 32'h1993;
  end else begin
    r <= r_next;
  end
end
```

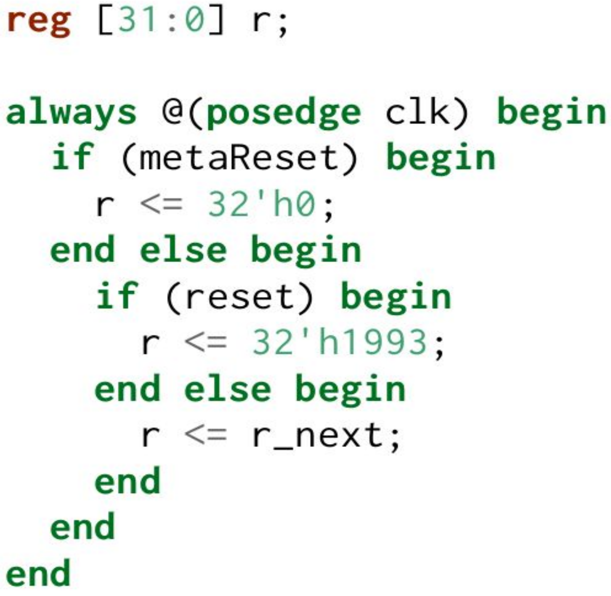
(a) Register With Reset

Meta Reset



```
reg [31:0] r;

always @(posedge clk) begin
  if (reset) begin
    r <= 32'h1993;
  end else begin
    r <= r_next;
  end
end
```

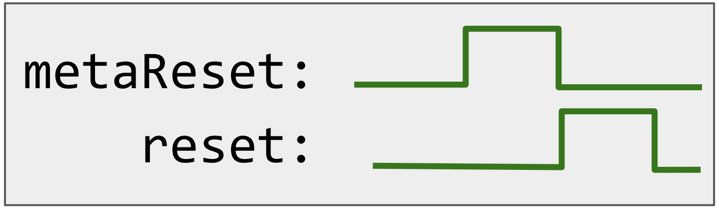
(a) Register With Reset

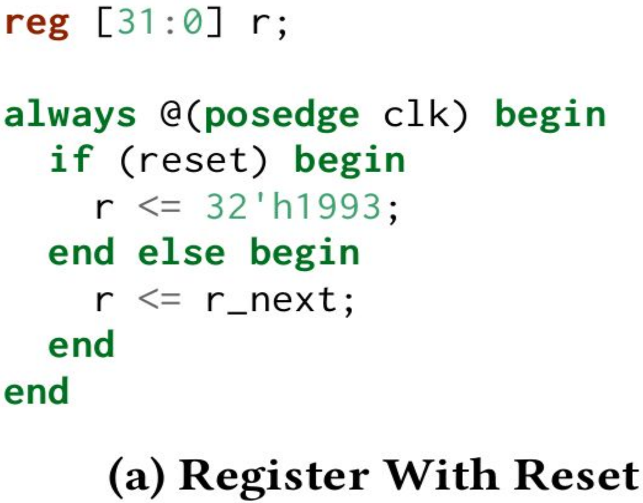


```
reg [31:0] r;

always @(posedge clk) begin
  if (metaReset) begin
    r <= 32'h0;
  end else begin
    if (reset) begin
      r <= 32'h1993;
    end else begin
      r <= r_next;
    end
  end
end
```

(b) Register With MetaReset

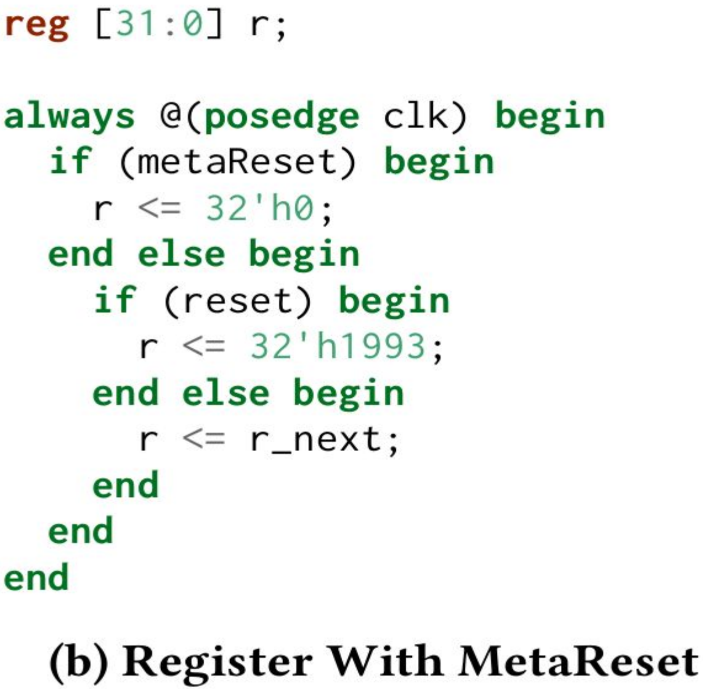
Meta Reset





```
reg [31:0] r;

always @(posedge clk) begin
  if (reset) begin
    r <= 32'h1993;
  end else begin
    r <= r_next;
  end
end
```

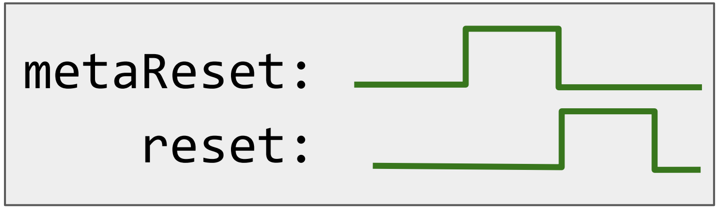
(a) Register With Reset

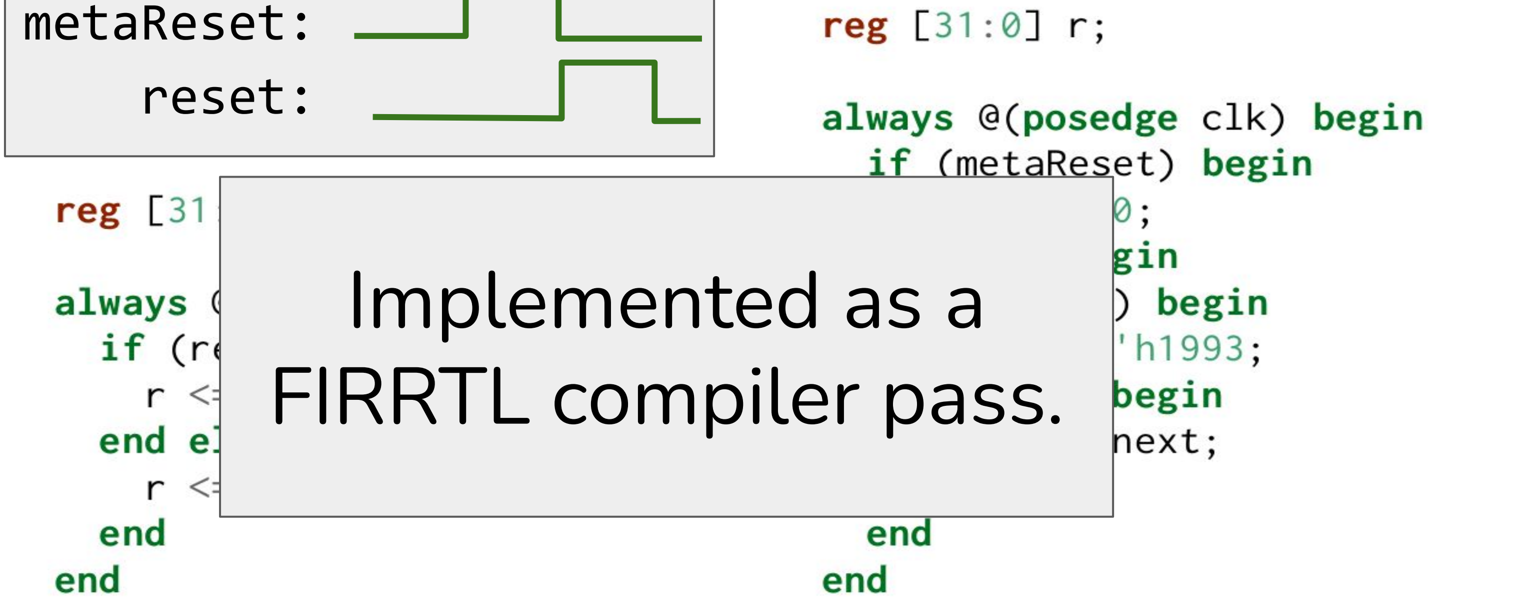


```
reg [31:0] r;

always @(posedge clk) begin
  if (metaReset) begin
    r <= 32'h0;
  end else begin
    if (reset) begin
      r <= 32'h1993;
    end else begin
      r <= r_next;
    end
  end
end
```

(b) Register With MetaReset

Meta Reset





```
reg [31

always @
  if (re
    r <=
  end el
    r <=
  end
end
```

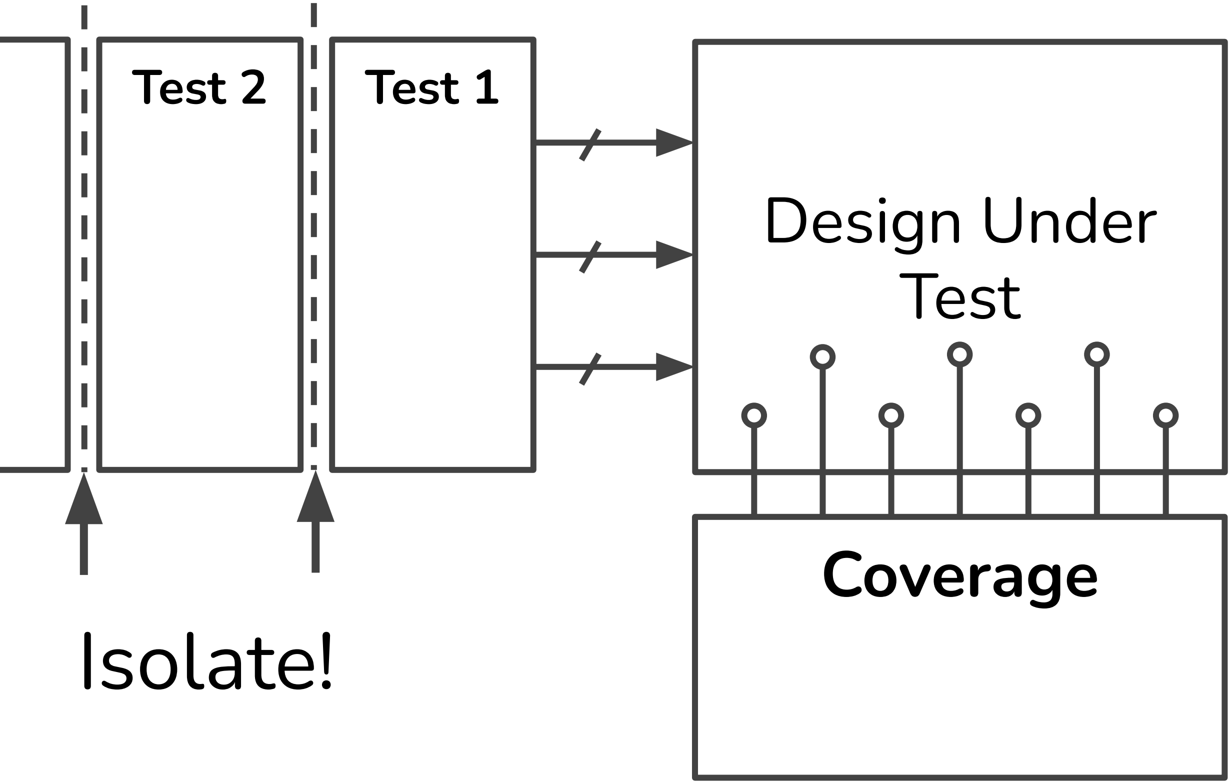
(a) Register With Reset

```
reg [31:0] r;

always @(posedge clk) begin
  if (metaReset) begin
                    0;
                   gin
                   ) begin
                   'h1993;
                   begin
                   next;

  end
end
```

(b) Register With MetaReset

Implemented as a FIRRTL compiler pass.

Deterministic Test Execution



Two Problems:

registers with undefined reset (XXXX)

memories

Sparse Memories

- Observation: short tests, < 100 cycles

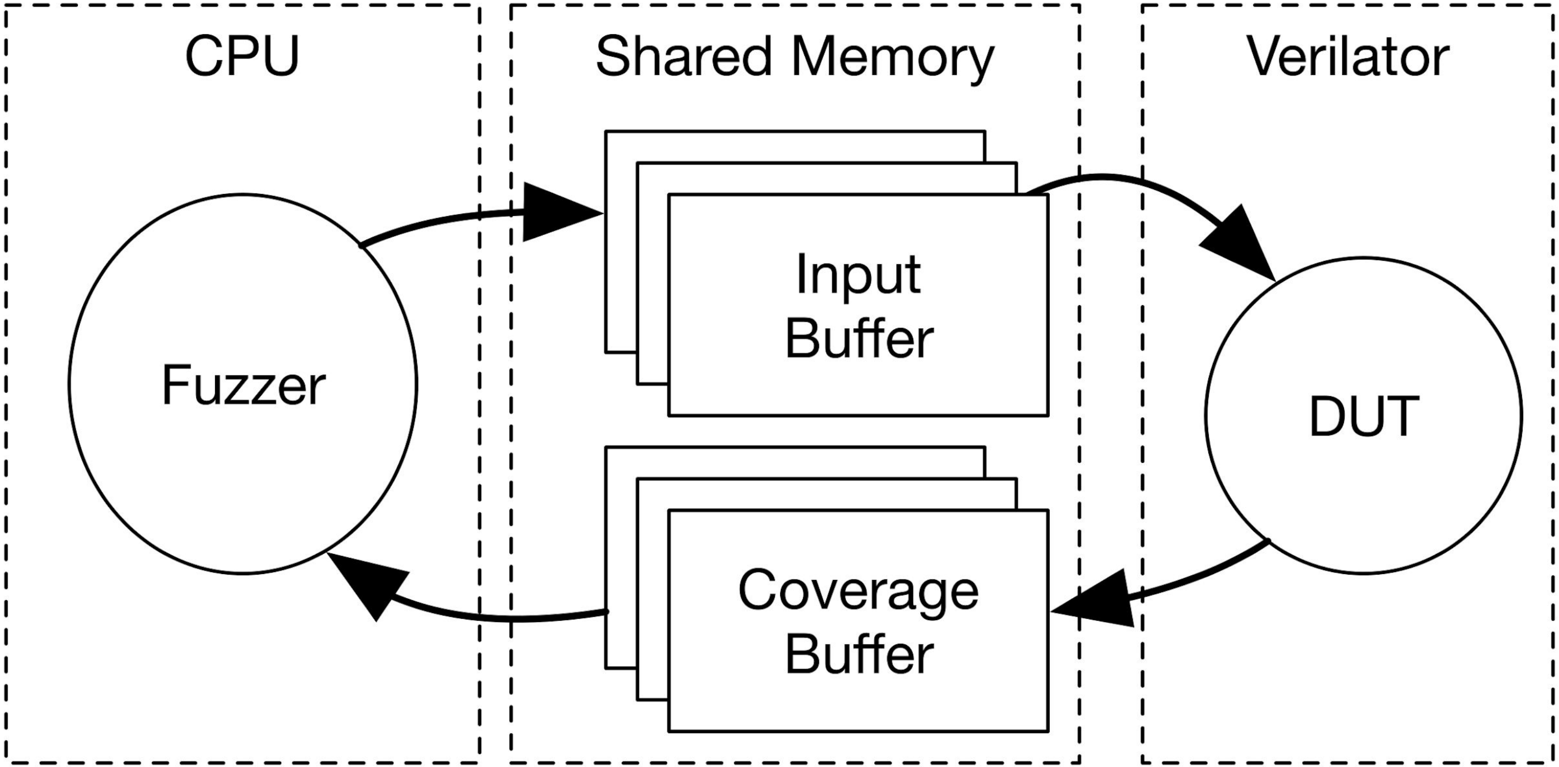
Sparse Memories

- Observation: short tests, < 100 cycles
- Number of memory writes bounded by #WritePorts x Cycles

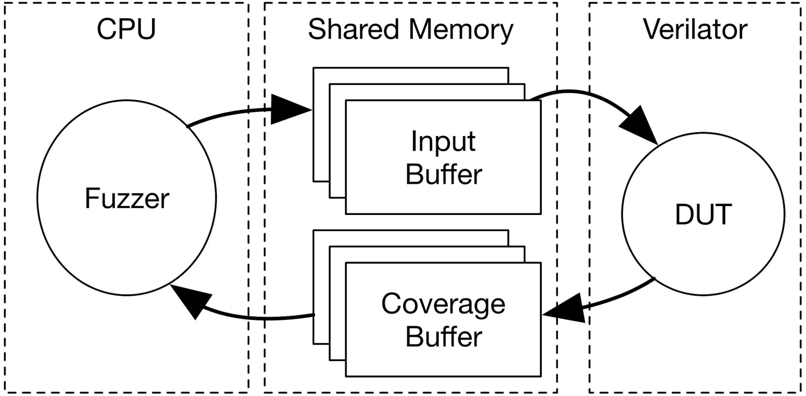
Sparse Memories

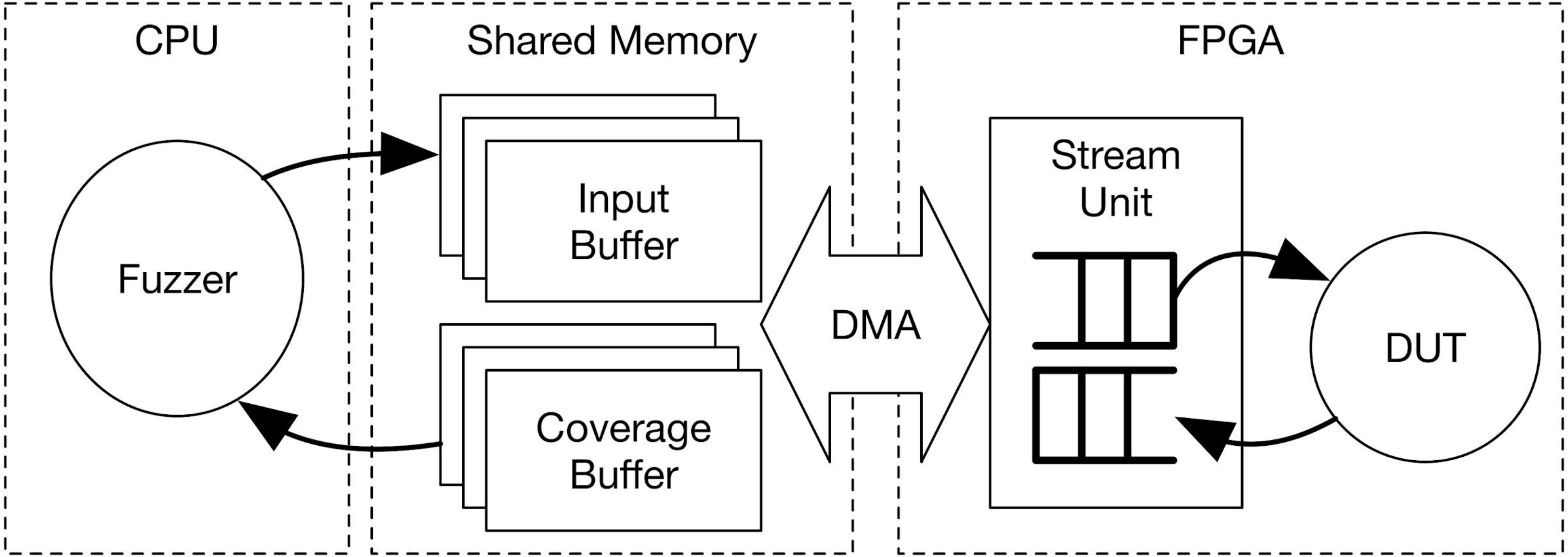
- Observation: short tests, < 100 cycles
- Number of memory writes bounded by #WritePorts x Cycles
- Sparse Memories:
    - use CAM to implement hardware hash table
    - reset in single cycle by setting valid bits to 0
    - FIRRTL compiler pass replaces all memories in the DUT with sufficiently large sparse memory implementation

Implementation

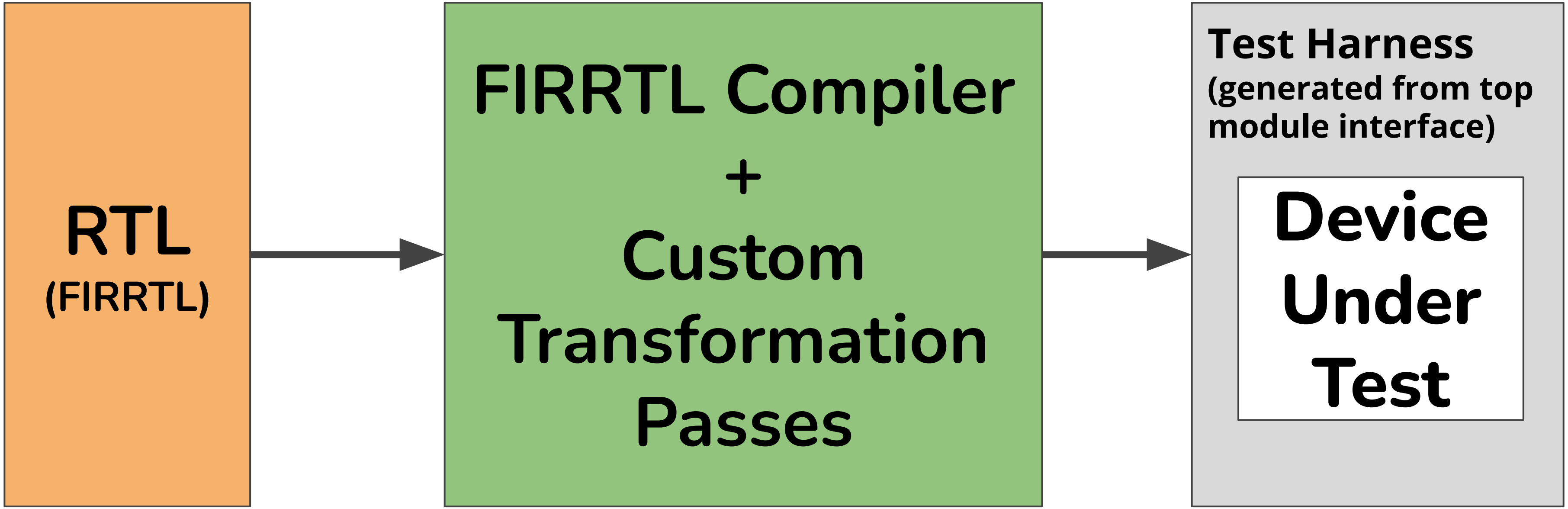


Implementation





Fully Automated Coverage Instrumentation and Harness Generation



RTL

(FIRRTL)

FIRRTL Compiler
+
Custom Transformation Passes

Test Harness
(generated from top module interface)

Device Under Test

Results

Results
1.) FPGA Speedup?

FPGA Emulation Speedup

	Sodor3Stage	Rocket
Lines of FIRRTL	4k	44k
Verilator	345 kHz	6.89 kHz
FPGA*	1.7 MHz	1.46 MHz
Speedup	4.9x	212x

FPGA Emulation Speedup

	Sodor3Stage	Rocket
Lines of FIRRTL	4k	44k
Verilator	345 kHz	6.89 kHz
FPGA*	1.7 MHz	1.46 MHz
Speedup	4.9x	212x

* Takes 2-3h for synthesis + place and route.

Results

1.) FPGA Speedup?

2.) Coverage Improvement?

Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore

Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore

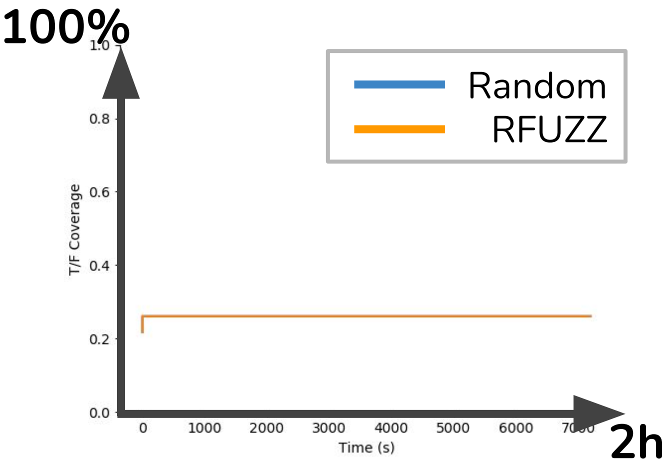
Coverage Results: Methodology

- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore
- Repeat experiments 4 times and average results

Coverage Results: Methodology

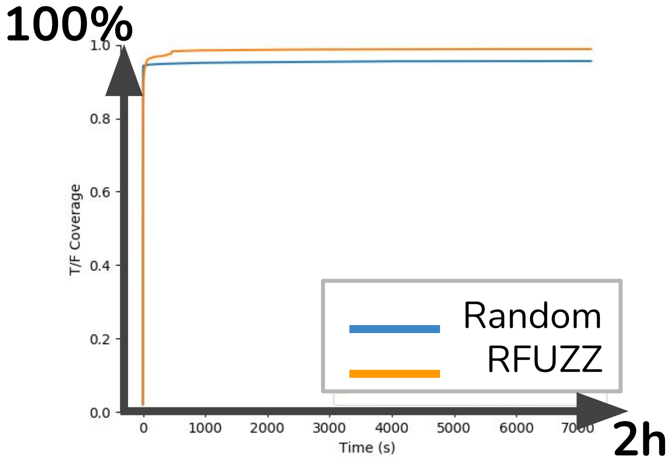
- Fuzz DUT for 2h on a single AWS vCore
- Generate random inputs for 2h on a single AWS vCore
- Repeat experiments 4 times and average results
- Graph **average mux toggle coverage** as a percentage of the maximum number of muxes in the DUT over time

Coverage Results



FFT

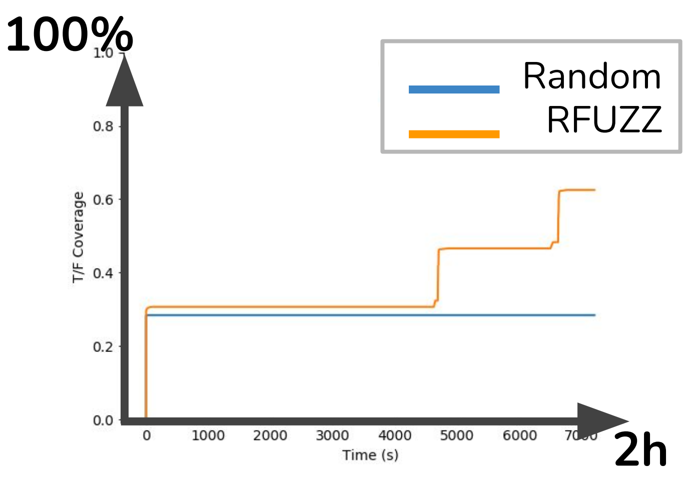
Lines of FIRRTL: 1545
Mux Cover Points: 195
Coverage Holes after Fuzzing: 85

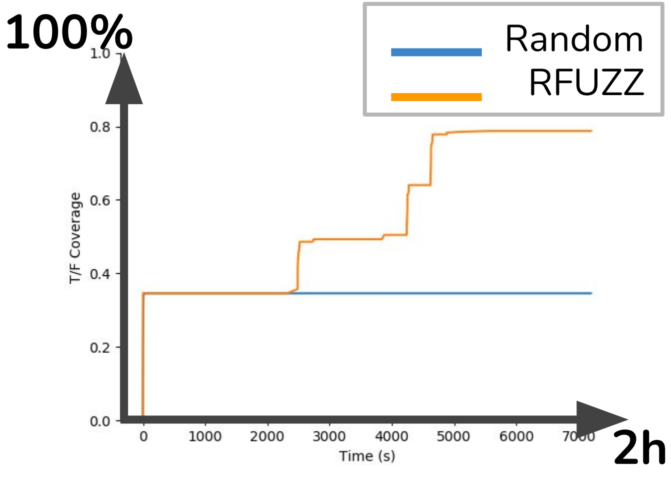


Sodor 3 Stage

Lines of FIRRTL: 4021
Mux Cover Points: 746
Coverage Holes after Fuzzing: 1-4

Coverage Results





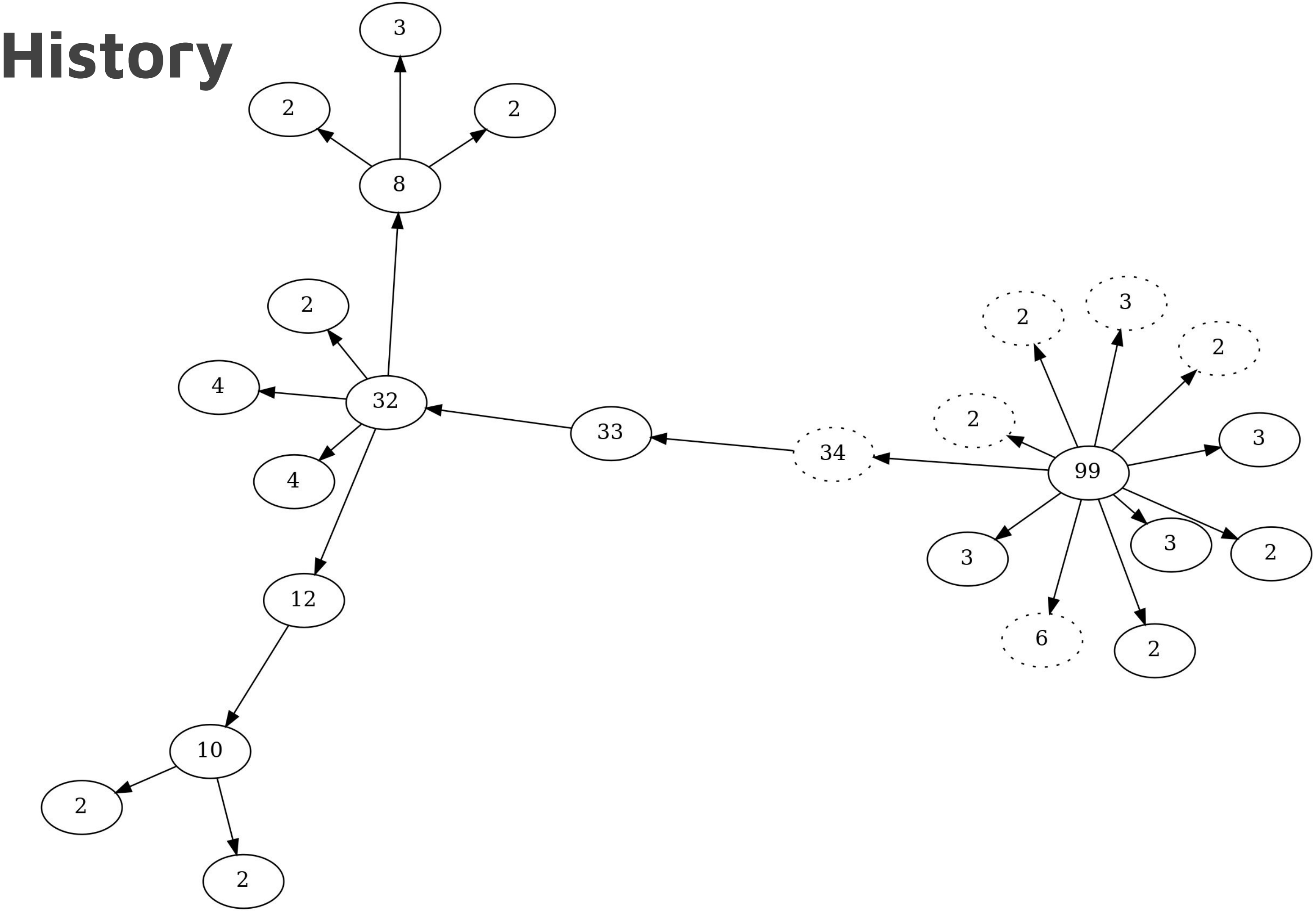
I2C

Lines of FIRRTL: 2373
Mux Cover Points: 301
Coverage Holes after Fuzzing: 5 - 61

SPI

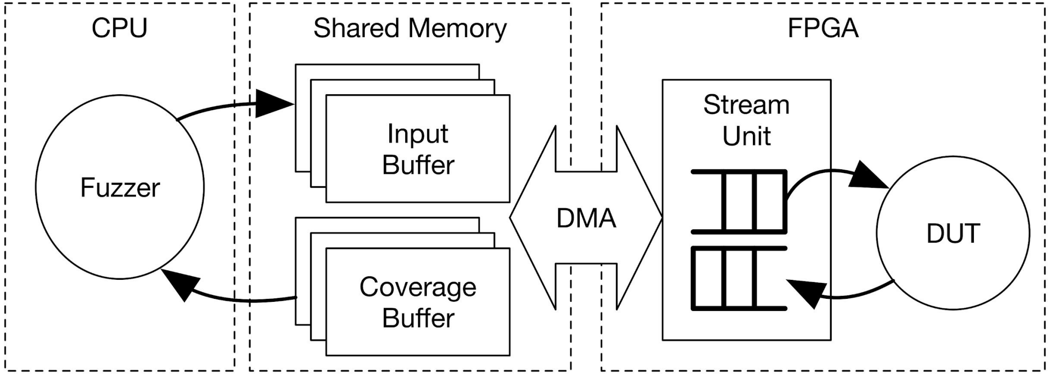
Lines of FIRRTL: 4046
Mux Cover Points: 323
Coverage Holes after Fuzzing: 7-70

Mutation History



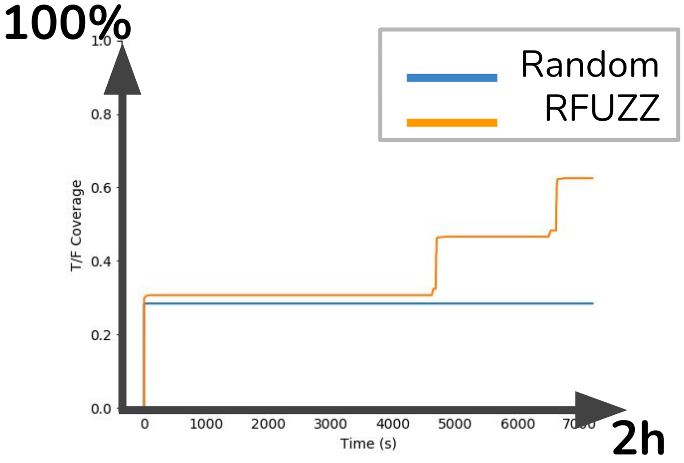
Thank you!

Questions?



Kevin Laeufer
laeufer@cs.berkeley.edu

<u>Reproduce + Extend our Results:</u>
github.com/ekiwi/rfuzz


T/F Coverage
Time (s)
2h
Random
RFUZZ