Hardware is the New Software: Finding Exploitable Bugs in Hardware Designs

Hardware Security @ UNC

Cynthia Sturton

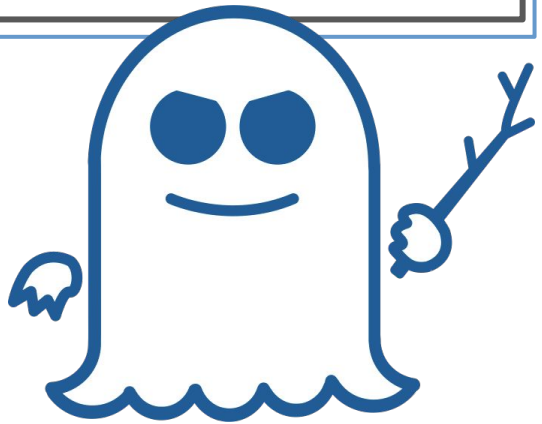
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Meltdown

Allows user applications to access operating system memory

Spectre

Almost every modern processor is affected



Foreshadow

Can expose the cryptographic keys that protect the integrity of SGX enclaves

[Date Prev][Date Next] [Thread Prev][Thread Next] [Date Index] [Thread Index]

[WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading

- *To*: debian-user@lists.debian.org, debian-devel@lists.debian.org
- *Subject*: [WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading
- *From*: Henrique de Moraes Holschuh <hmh@debian.org>
- *Date*: Sun, 25 Jun 2017 09:19:36 -0300

The Cyrix 6x86 Coma Bug

The Pentium F00F Bug

by Robert R. Collins

Hypervisor headaches: Hosts hosed by x86 exception bugs

Microsoft, Xen, KVM *et al* need patches

By Richard Chirgwin 13 Nov 2015 at 04:56 16 SHARE ▼

Various hypervisors and operating systems are scrambling to patch

[Date Prev][Date Next] [Thread Prev][Thread Next] [Date Index] [Thread Index]

[WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading

- *To*: debian-user@lists.debian.org, debian-devel@lists.debian.org
- *Subject*: [WARNING] Intel Skylake/Kaby Lake processors: broken hyper-threading
- *From*: Henrique de Moraes Holschuh <hmh@debian.org>
- *Date*: Sun, 25 Jun 2017 09:19:36 -0300

The Cyrix 6x86 Coma Bug

The Pentium F00F Bug
by Robert R. Collins

1997

1998

Hypervisor headaches: Hosts hosed by x86 exception bugs

Microsoft, Xen, KVM *et al* need patches

By Richard Chirgwin 13 Nov 2015 at 04:56 16 SHARE ▼

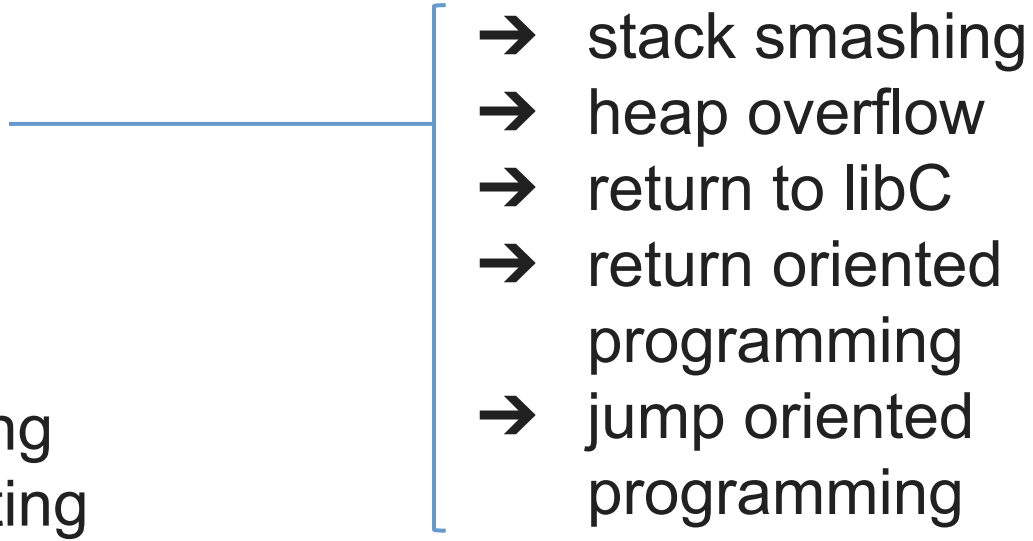
Various hypervisors and operating systems are scrambling to patch

Software Security

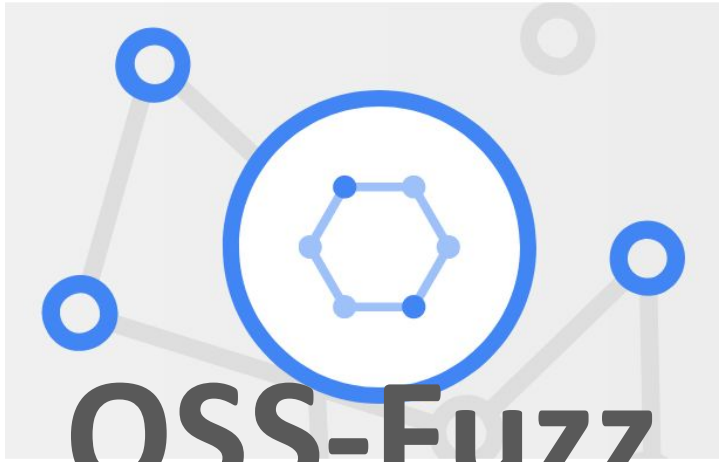
- Buffer overflow
- Integer overflow
- Format string
- SQL injection
- Directory crawling
- Cross-site scripting
- Cross-site request forgery

Software Security

- Buffer overflow
- Integer overflow
- Format string
- SQL injection
- Directory crawling
- Cross-site scripting
- Cross-site request forgery

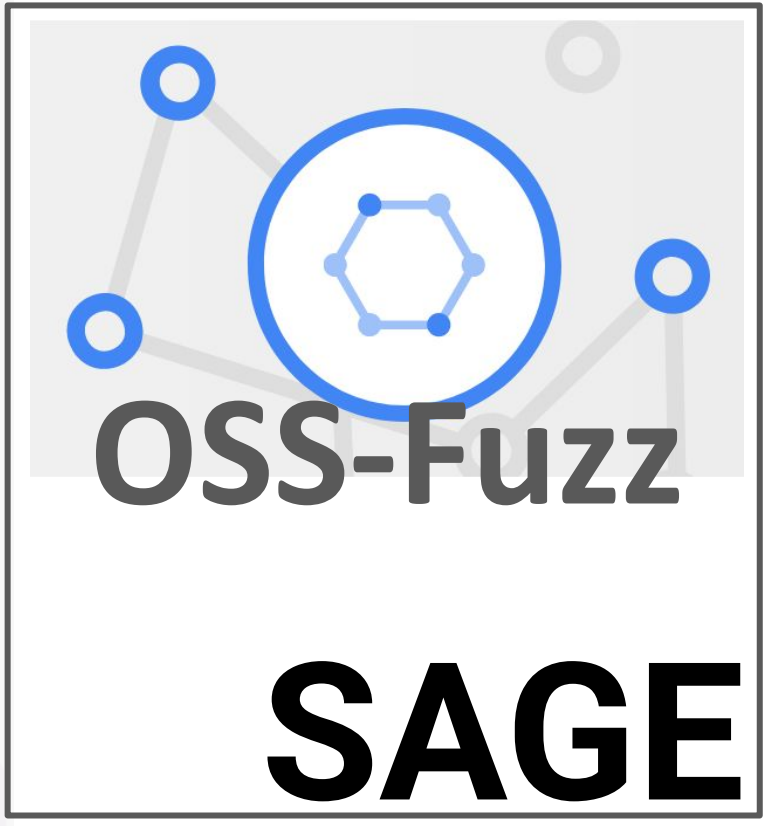


- ➔ stack smashing
- ➔ heap overflow
- ➔ return to libC
- ➔ return oriented programming
- ➔ jump oriented programming



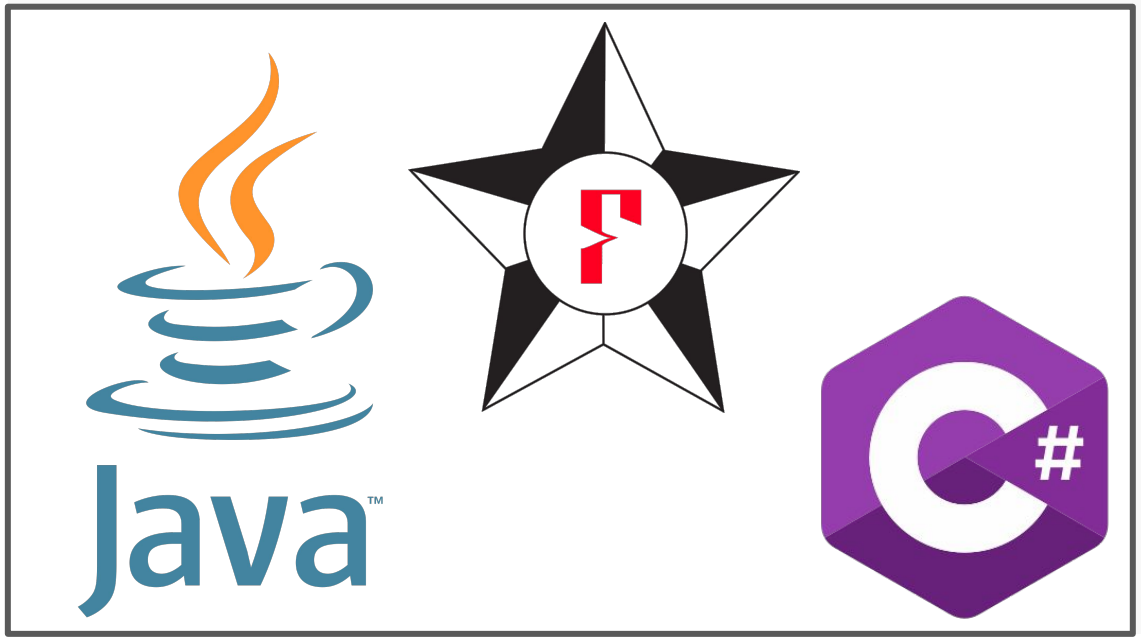
SAGE

fuzzing



program analysis

secure languages



Hardware Security

- Secure languages
- Manual review

Hardware Security

- Side channels
- Transient faults

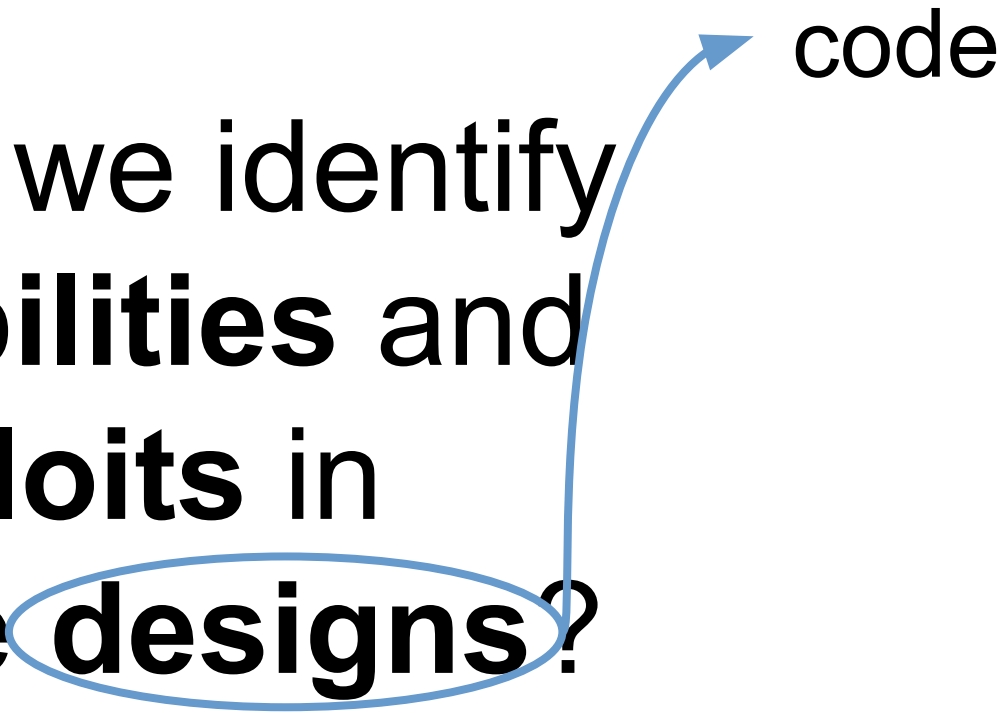
How can we identify **vulnerabilities** and their **exploits** in hardware **designs**?

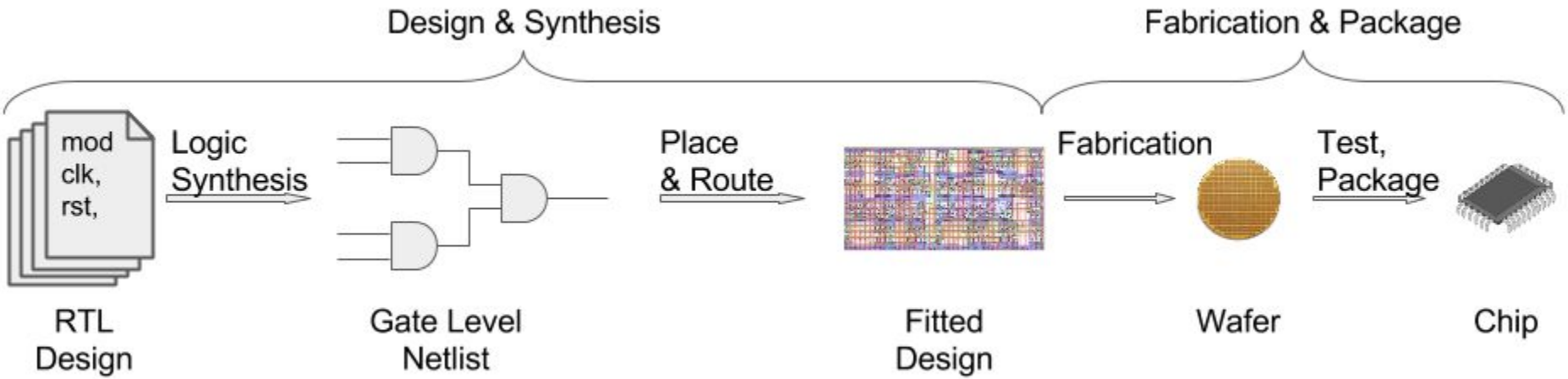
How can we identify **vulnerabilities** and their **exploits** in hardware **designs**?

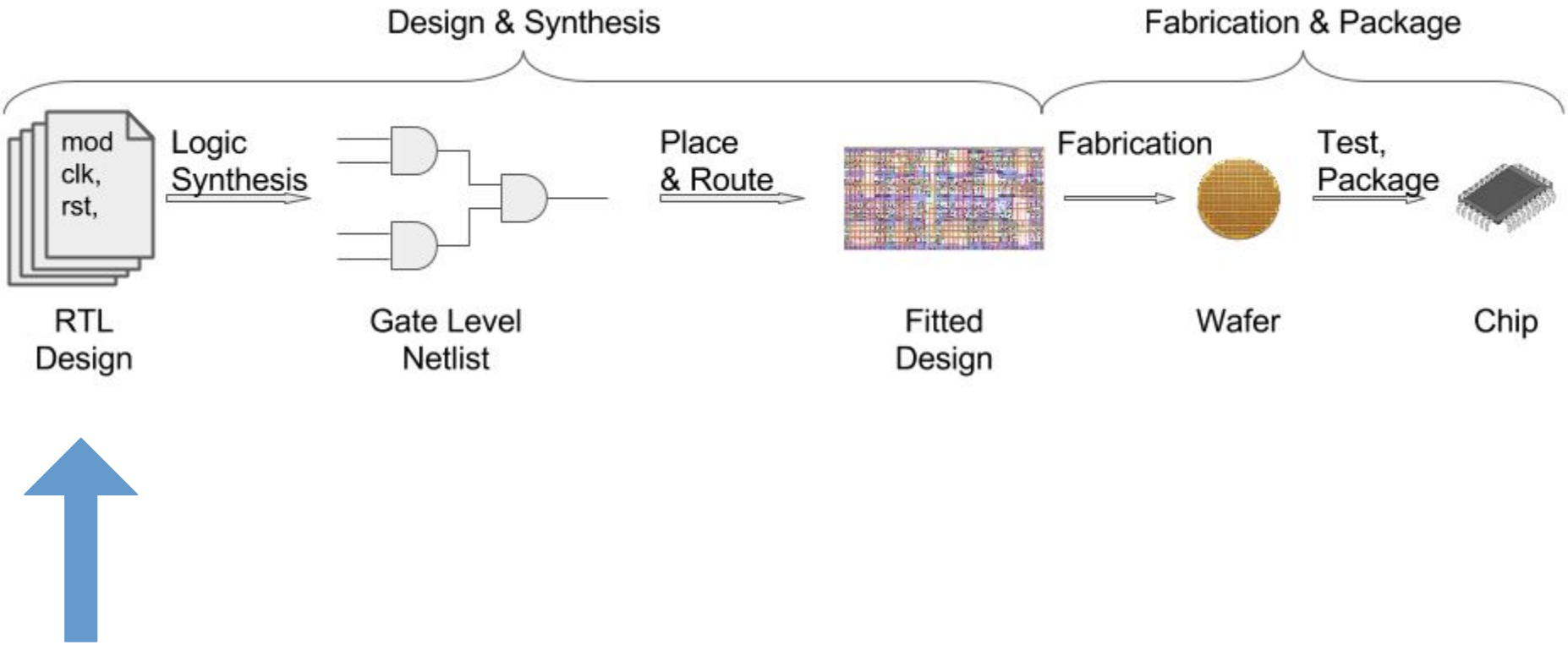
property violations

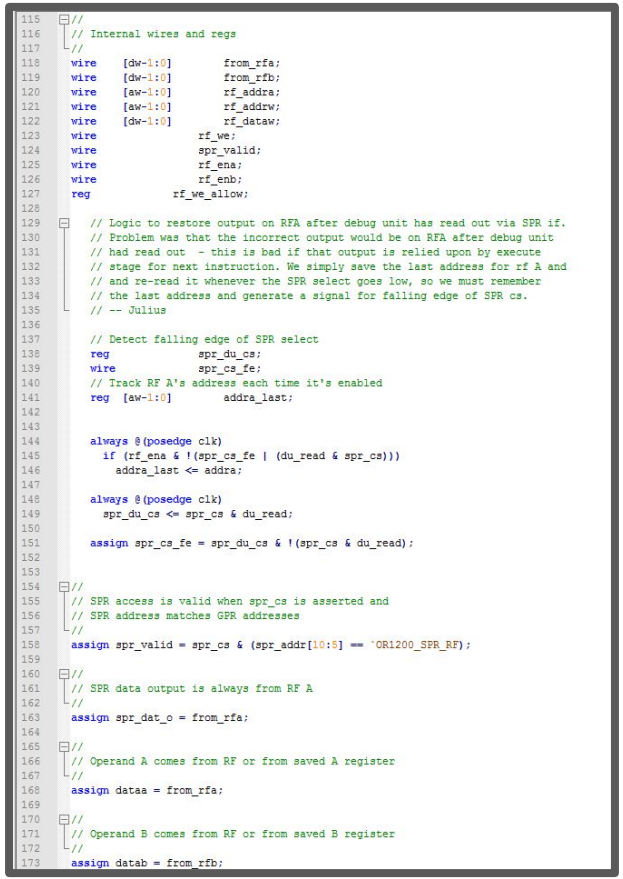
How can we identify **vulnerabilities** and their **exploits** in hardware **designs**?

How can we identify **vulnerabilities** and their **exploits** in hardware **designs**?




Design & Synthesis
Fabrication & Package
mod
clk,
rst,
Logic
Synthesis
Place
& Route
Fabrication
Test,
Package
RTL
Design
Gate Level
Netlist
Fitted
Design
Wafer
Chip

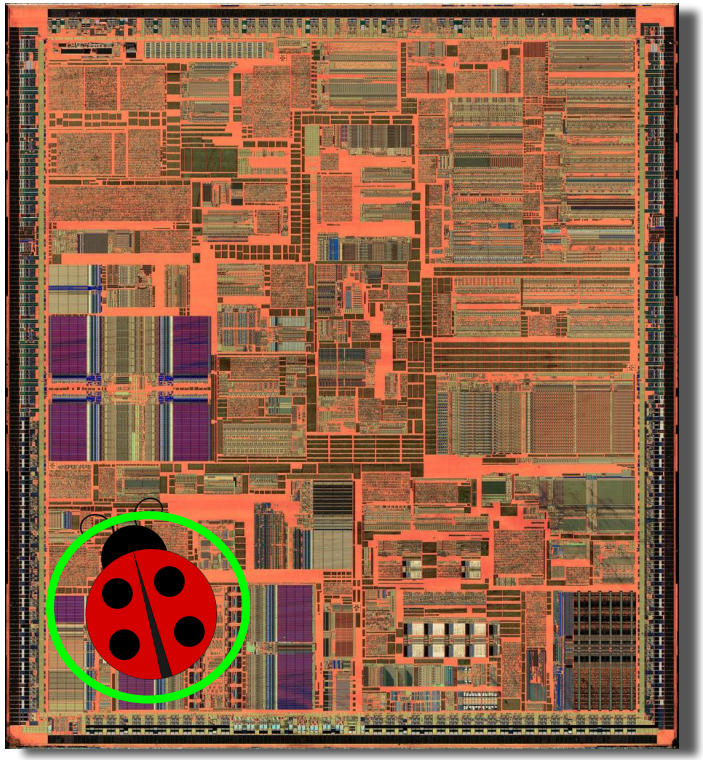

Design & Synthesis
Fabrication & Package
mod
clk,
rst,
Logic
Synthesis
Place
& Route
Fabrication
Test,
Package
RTL
Design
Gate Level
Netlist
Fitted
Design
Wafer
Chip



```
115  //
116  // Internal wires and regs
117  //
118  wire    [dw-1:0]        from_rfa;
119  wire    [dw-1:0]        from_rfb;
120  wire    [aw-1:0]        rf_addra;
121  wire    [aw-1:0]        rf_addrw;
122  wire    [dw-1:0]        rf_dataw;
123  wire                rf_we;
124  wire                spr_valid;
125  wire                rf_ena;
126  wire                rf_enb;
127  reg             rf_we_allow;
128
129    // Logic to restore output on RFA after debug unit has read out via SPR if.
130    // Problem was that the incorrect output would be on RFA after debug unit
131    // had read out  - this is bad if that output is relied upon by execute
132    // stage for next instruction. We simply save the last address for rf A and
133    // and re-read it whenever the SPR select goes low, so we must remember
134    // the last address and generate a signal for falling edge of SPR cs.
135    // -- Julius
136
137    // Detect falling edge of SPR select
138    reg             spr_du_cs;
139    wire            spr_cs_fe;
140    // Track RF A's address each time it's enabled
141    reg  [aw-1:0]       addra_last;
142
143
144    always @(posedge clk)
145      if (rf_ena & !(spr_cs_fe | (du_read & spr_cs)))
146        addra_last <= addra;
147
148    always @(posedge clk)
149      spr_du_cs <= spr_cs & du_read;
150
151    assign spr_cs_fe = spr_du_cs & !(spr_cs & du_read);
152
153
154  //
155  // SPR access is valid when spr_cs is asserted and
156  // SPR address matches GPR addresses
157  //
158  assign spr_valid = spr_cs & (spr_addr[10:5] == `OR1200_SPR_RF);
159
160  //
161  // SPR data output is always from RF A
162  //
163  assign spr_dat_o = from_rfa;
164
165  //
166  // Operand A comes from RF or from saved A register
167  //
168  assign dataa = from_rfa;
169
170  //
171  // Operand B comes from RF or from saved B register
172  //
173  assign datab = from_rfb;
```

Vulnerabilities: An Analysis of Exploitable Bugs

[ASPLOS 2015]



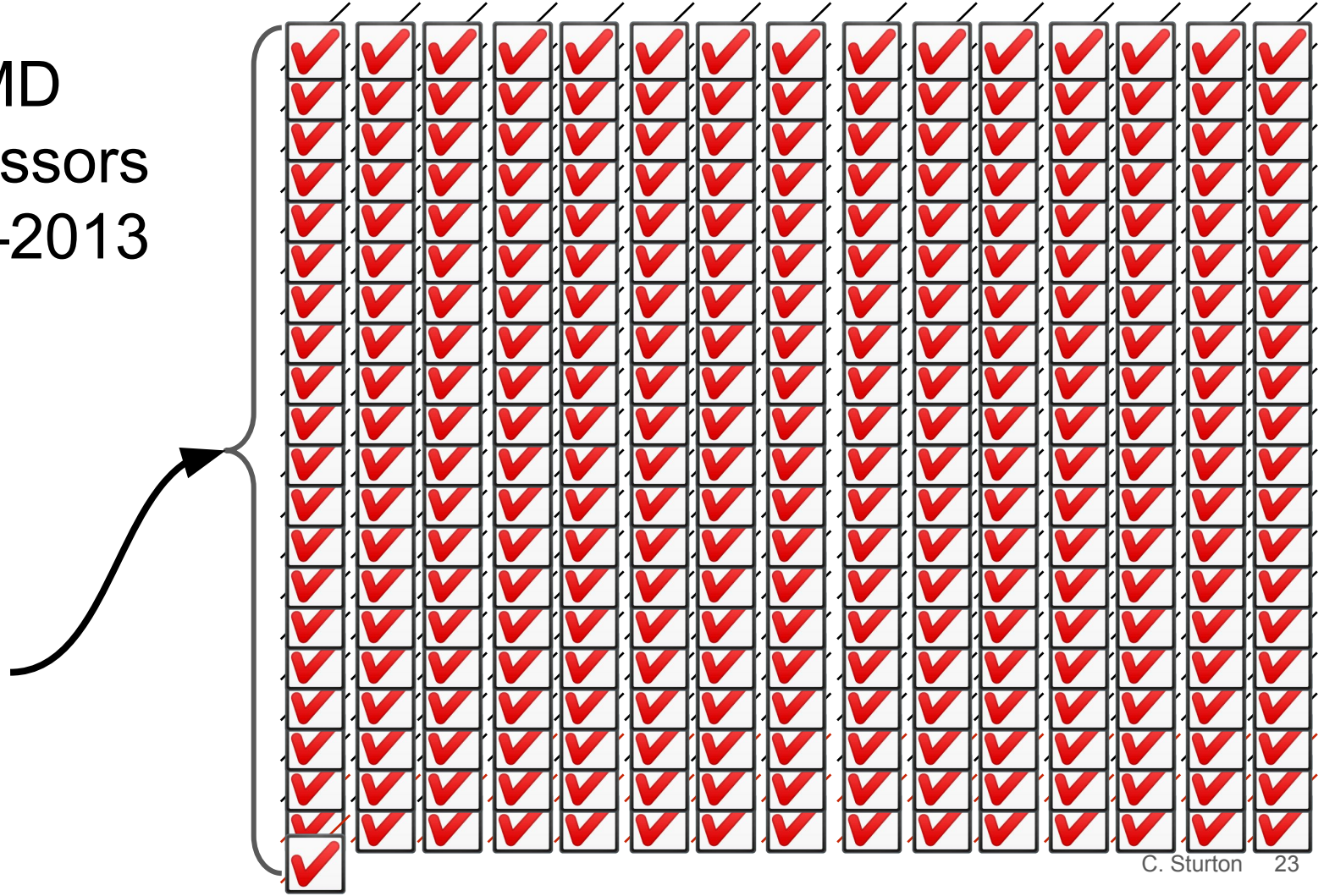
AMD Errata #776

Incorrect Processor Branch Prediction for Two Consecutive Linear Pages

Under a highly specific and detailed set of internal timing conditions, the processor core may incorrectly fetch instructions ...

Potential effect: **unpredictable system behavior**

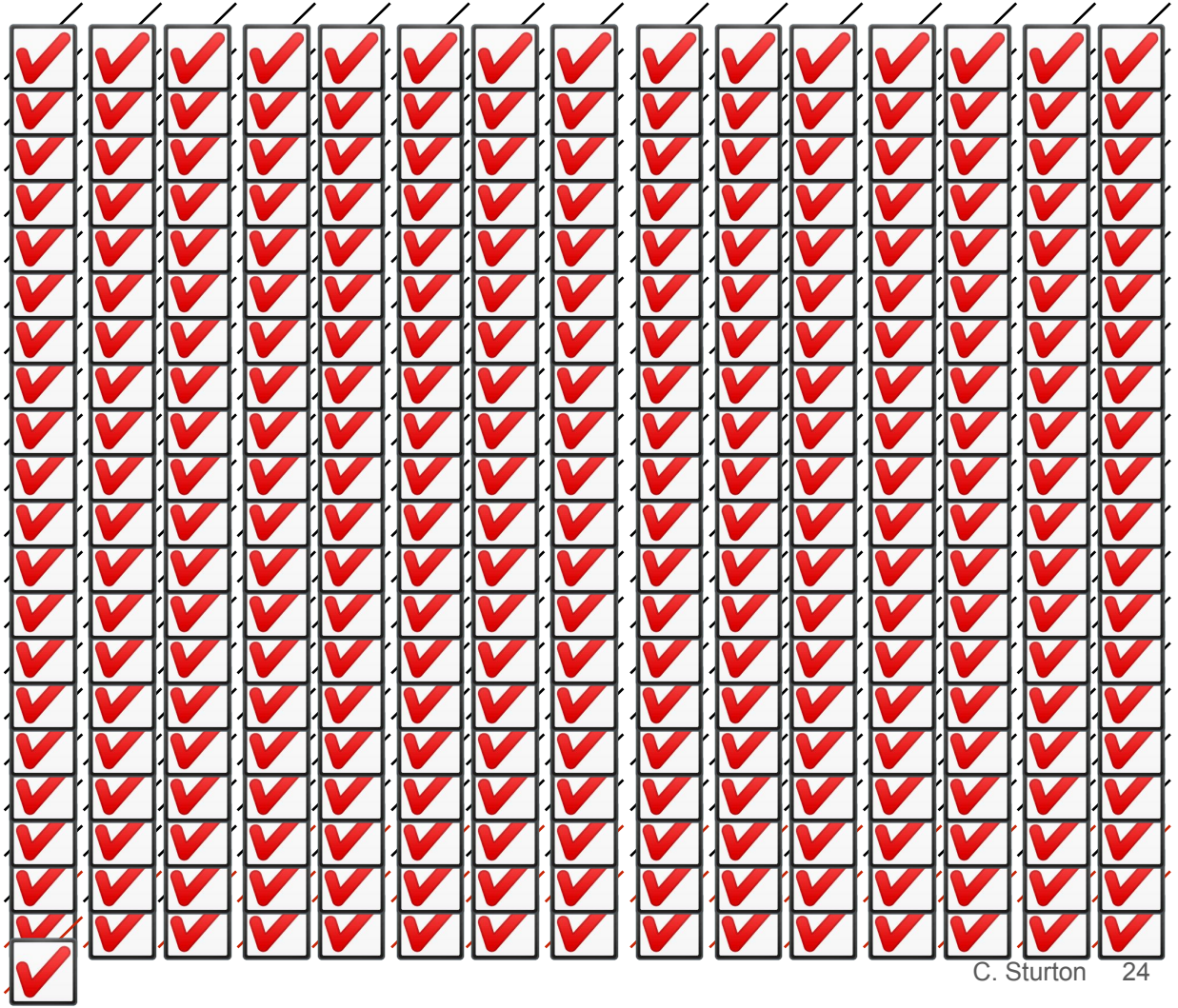
AMD Processors 2007–2013



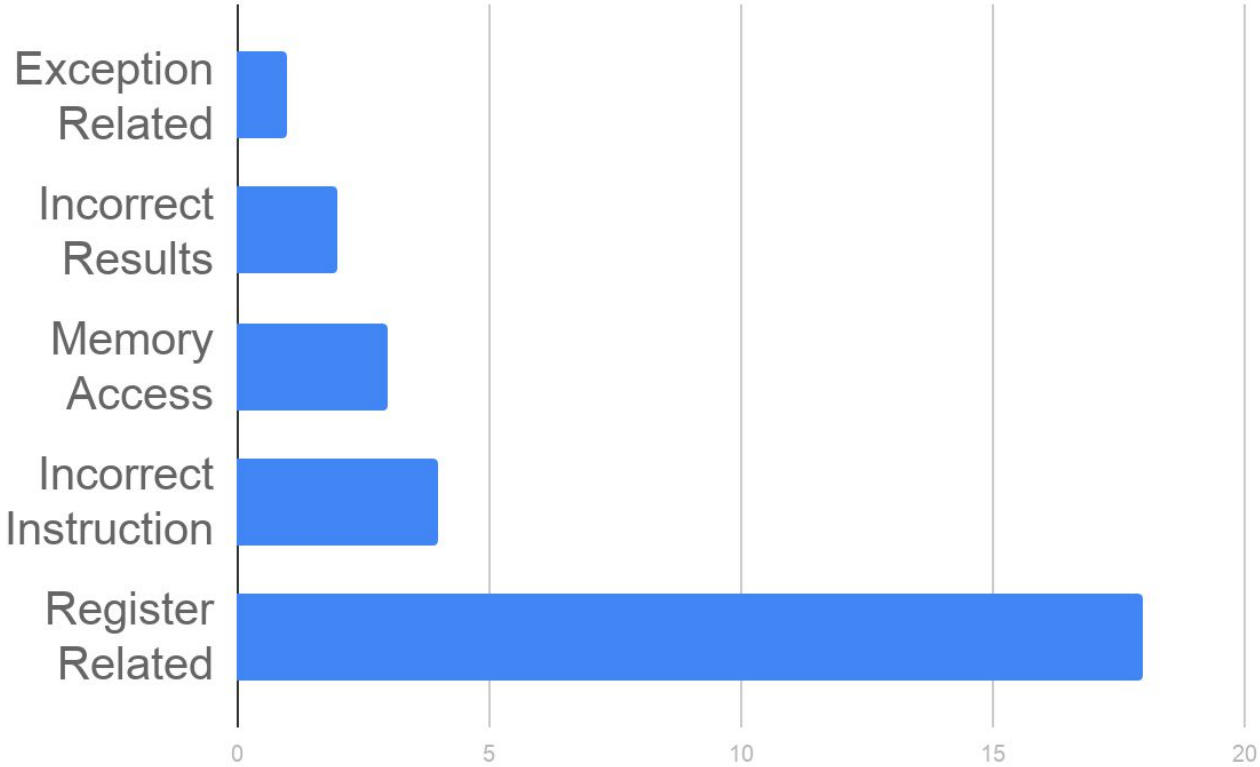
301 errata

AMD Processors 2007–2013

28 security critical



Classifying Exploitable Bugs



Manually Writing Security Properties

[ASPLOS 2015]

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ
φ φ φ φ φ φ φ

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ
φ φ φ φ φ φ φ

φ: Processor mode changes from low privilege to high privilege only by an exception or a reset.

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ
φ φ φ φ φ φ φ

AMD Errata (3): φ φ φ

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ φ φ φ φ φ φ φ φ

AMD Errata (3): φ φ φ

φ: When a register changes, it must be specified as the target of the instruction.

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ φ φ φ φ φ φ φ φ

AMD Errata (3): φ φ φ

Initial Evaluation (1): φ

Writing Security Properties

Specification Documents (14): φ φ φ φ φ φ φ
φ φ φ φ φ φ φ

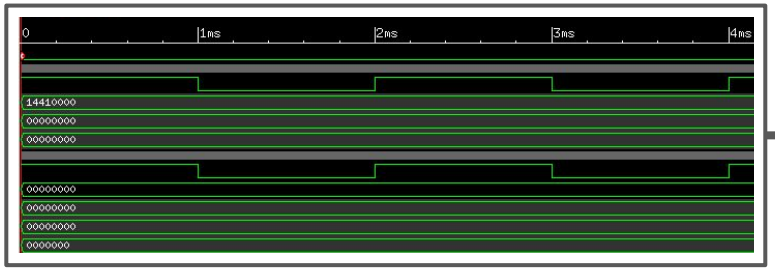
AMD Errata (3): φ φ φ

Initial Evaluation (1): φ

φ: The instruction does not change in the pipeline.

Security Property Specification

[ASPLOS 2017]



Traces

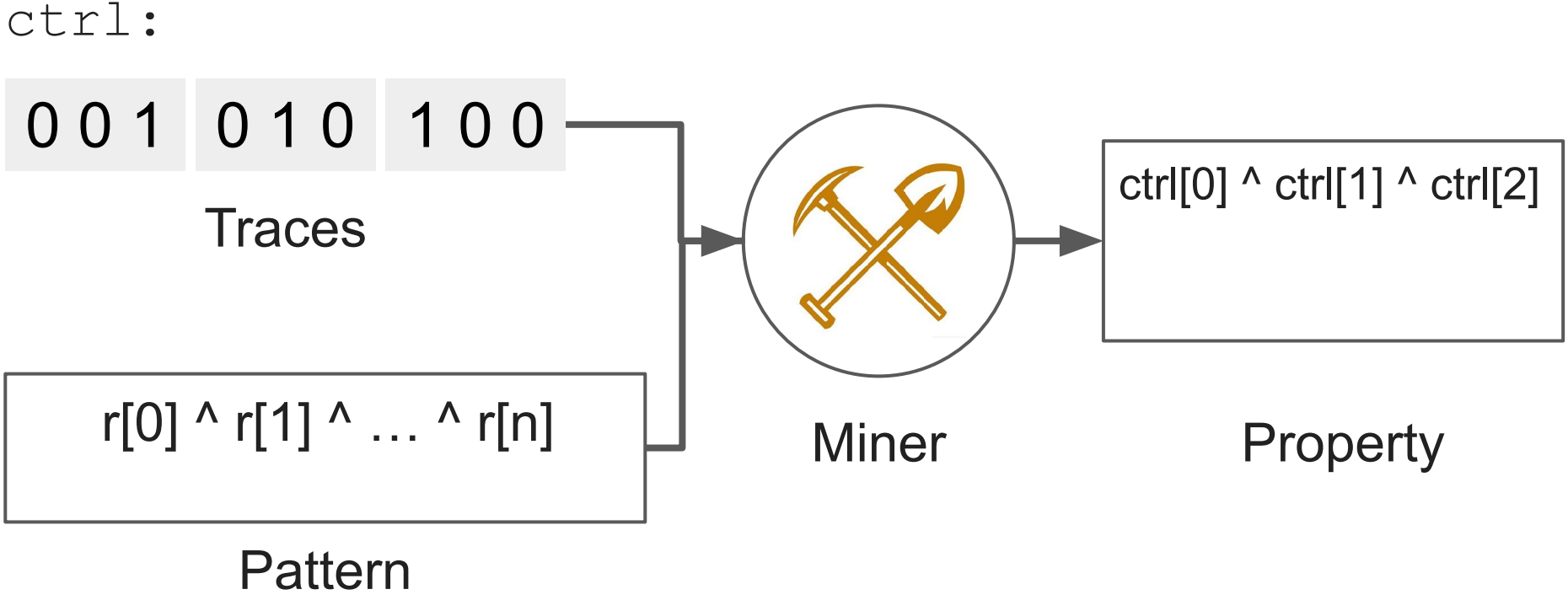
- p_1
- p_2
- ...

Patterns

Miner

- φ_1
- φ_2
- ...
- φ_n

Properties

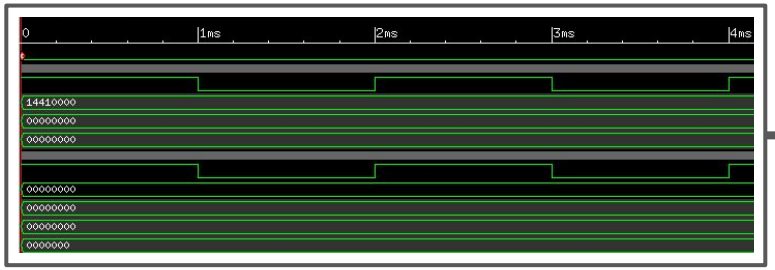

ctrl:
0 0 1
0 1 0
1 0 0
Traces
r[0] ^ r[1] ^ … ^ r[n]
Pattern
Miner
ctrl[0] ^ ctrl[1] ^ ctrl[2]
Property

Example of Exploitable Bug

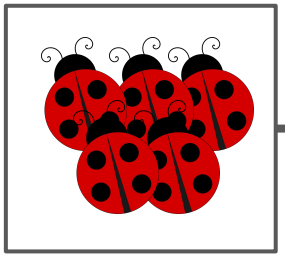
```
assign a_lt_b = comp_op[3] ? ((a[width - 1] & !b[width - 1]) |
    (!a[width - 1] & !b[width - 1] & result_sum[width - 1]) |
    (a[width - 1] & b[width - 1] & result_sum[width - 1])) :
    result_sum[width - 1];
```

Example of Exploitable Bug

```
assign a_lt_b = comp_op[3] ? ((a[width - 1] & !b[width - 1]) |
    (!a[width - 1] & !b[width - 1] & result_sum[width - 1]) |
    (a[width - 1] & b[width - 1] & result_sum[width - 1])) :
    result_sum[width - 1];
    (a < b);
```



Traces



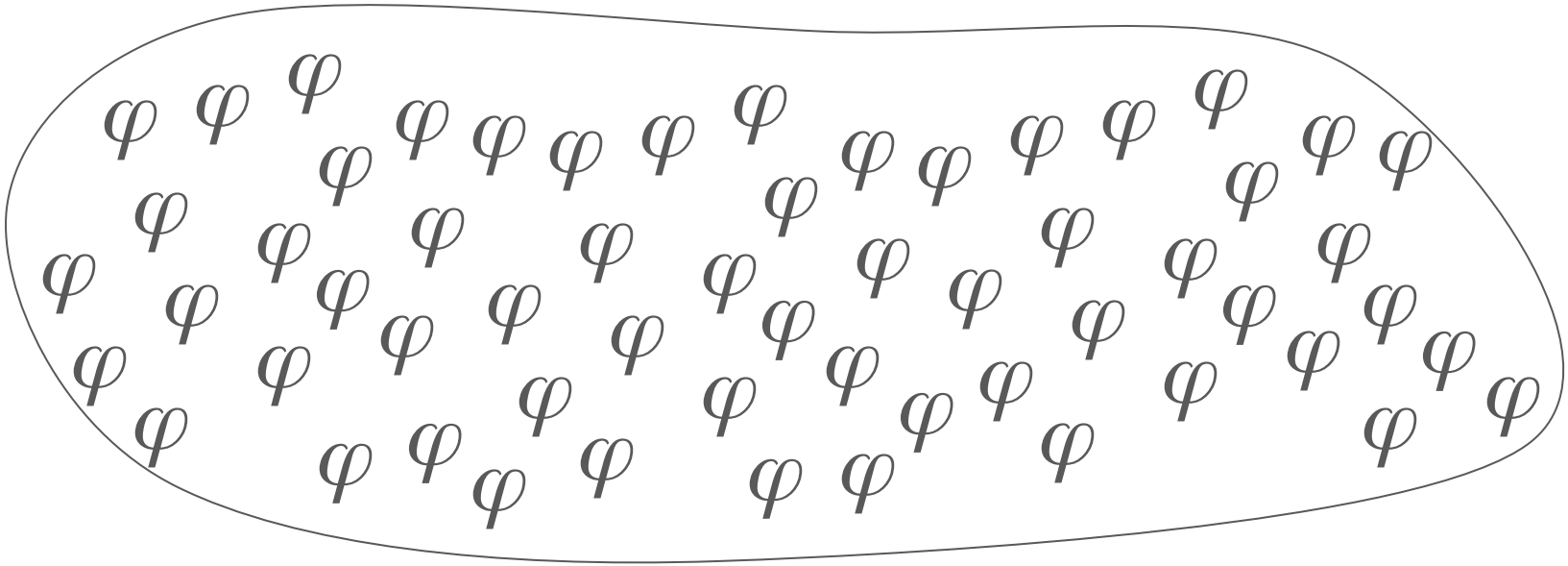
Security Errata

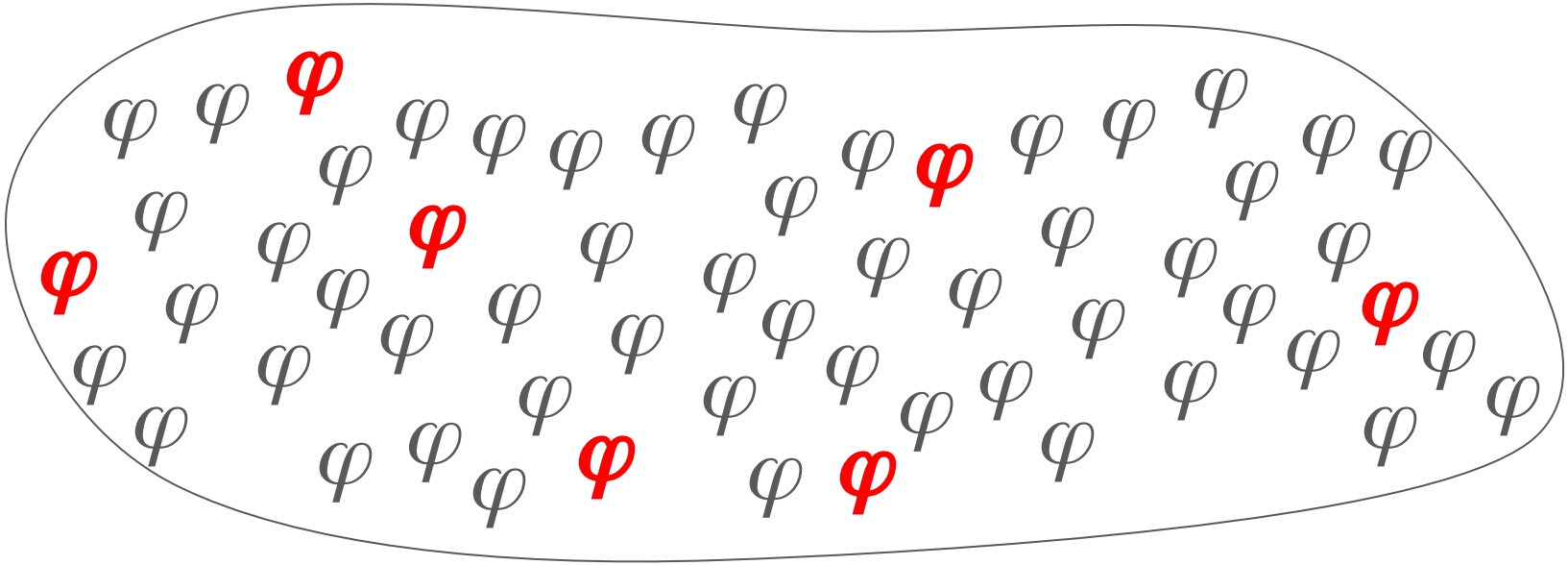
SCI Finder

Miner

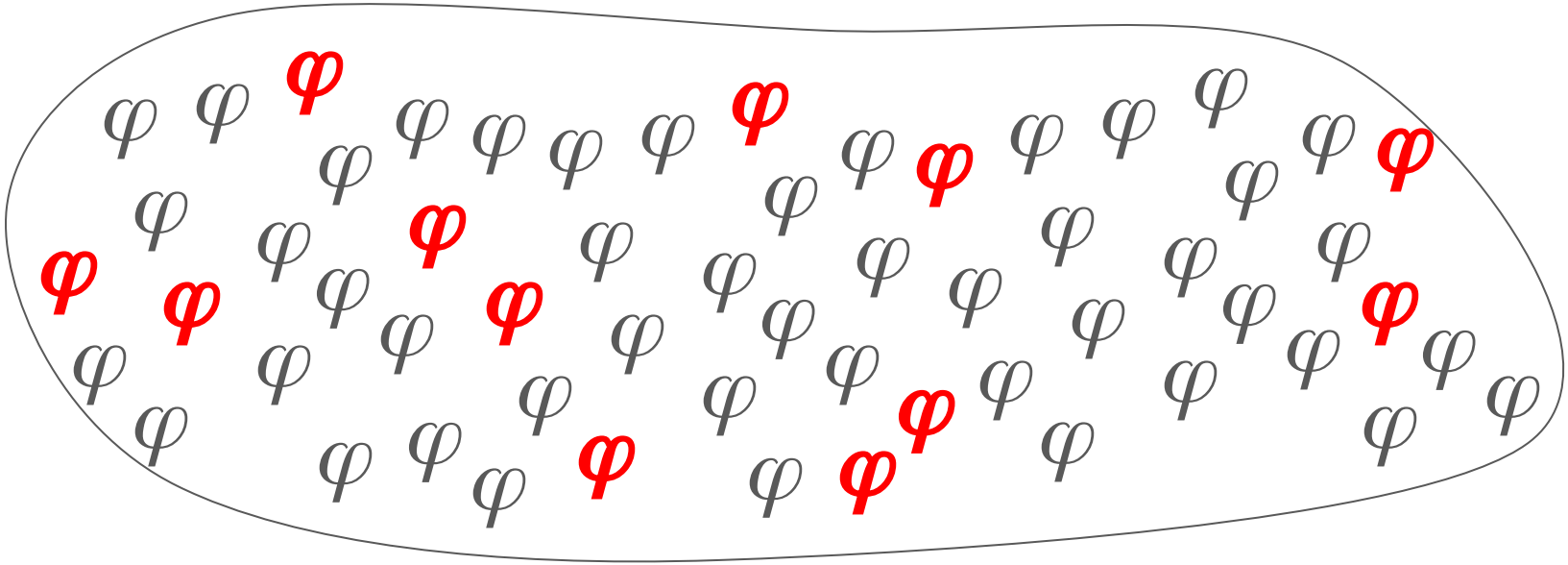
- φ_1
- φ_2
- ...
- φ_n

Properties



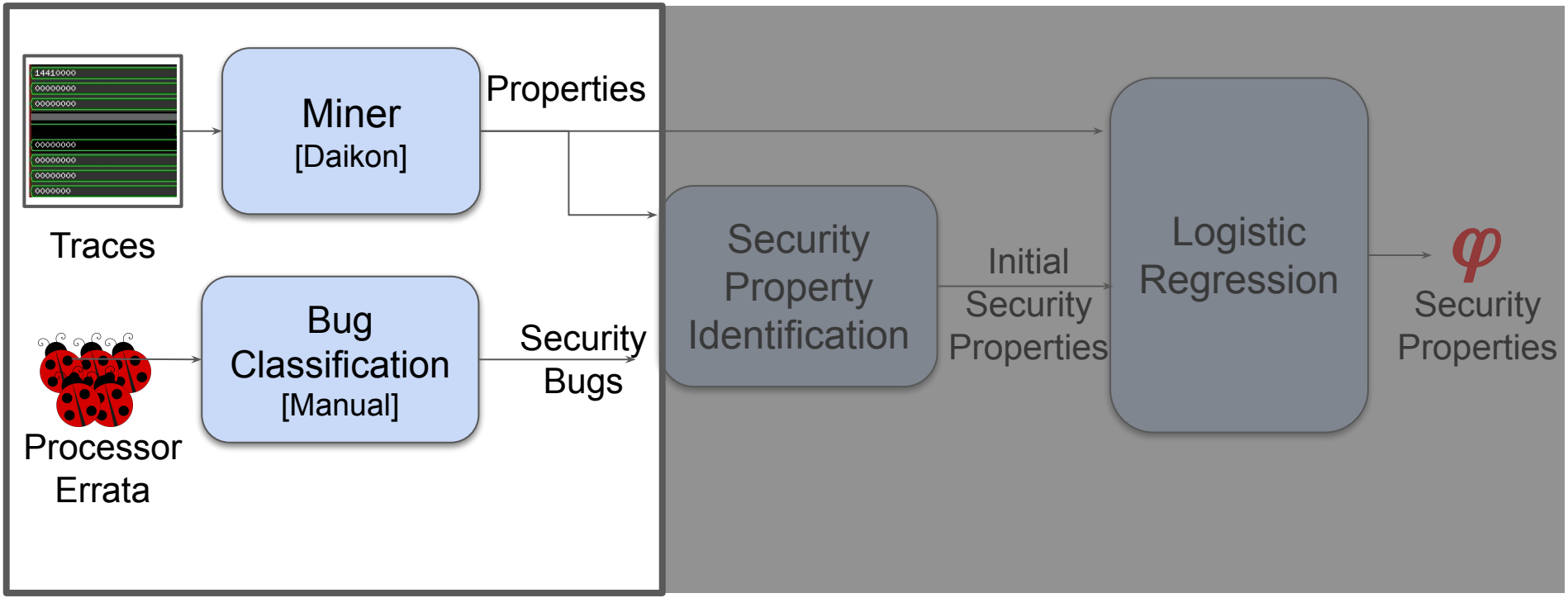


1. Known bugs → Demonstrate exploit → Security properties

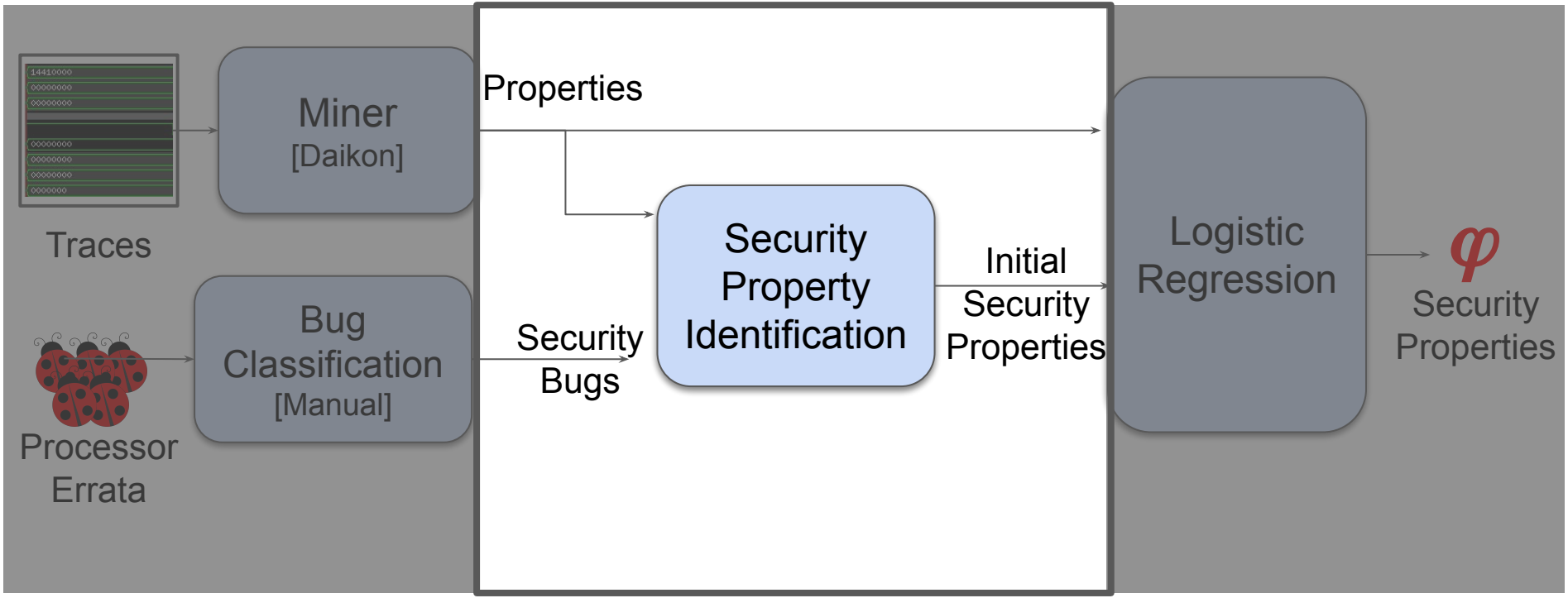


1. Known bugs → Demonstrate exploit → Security properties
2. Machine learning → Additional security properties

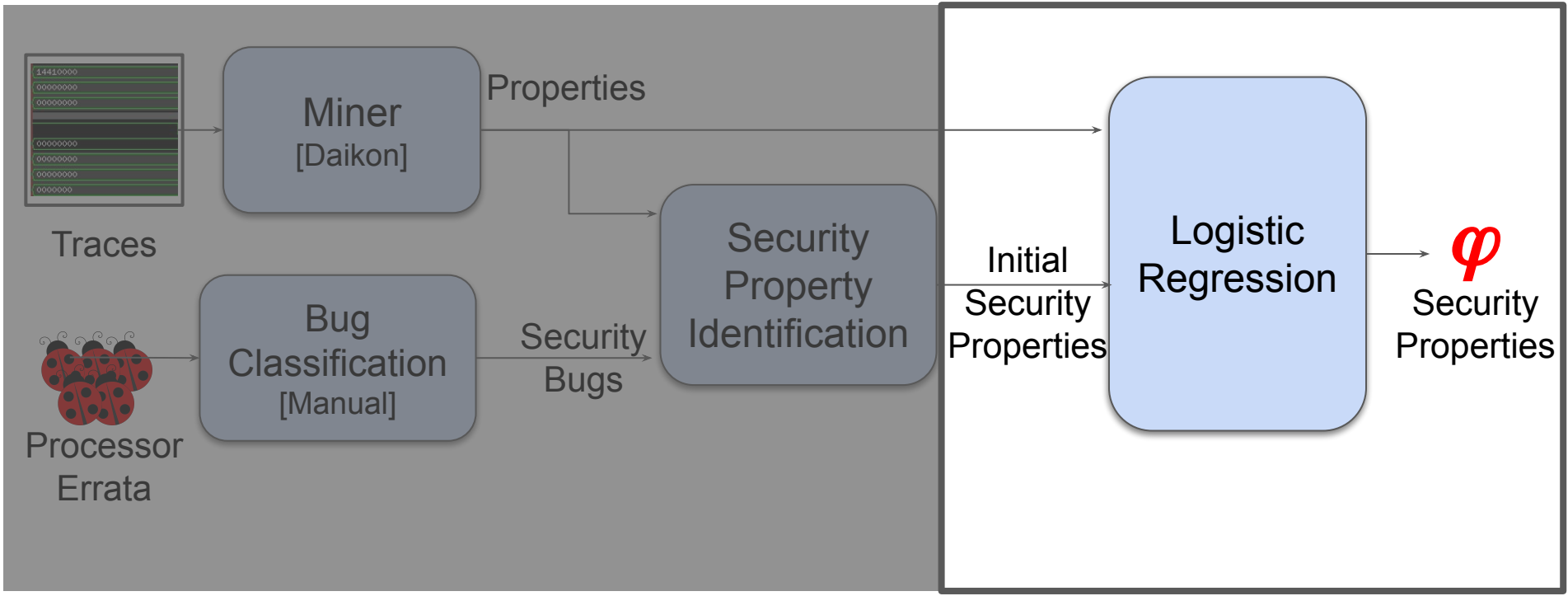
SCIFinder


Traces
Miner
[Daikon]
Properties
Processor
Errata
Bug
Classification
[Manual]
Security
Bugs
Security
Property
Identification
Initial
Security
Properties
Logistic
Regression
φ
Security
Properties

SCIFinder


Traces
Miner
[Daikon]
Processor
Errata
Bug
Classification
[Manual]
Properties
Security
Bugs
Security
Property
Identification
Initial
Security
Properties
Logistic
Regression
Security
Properties

SCIFinder

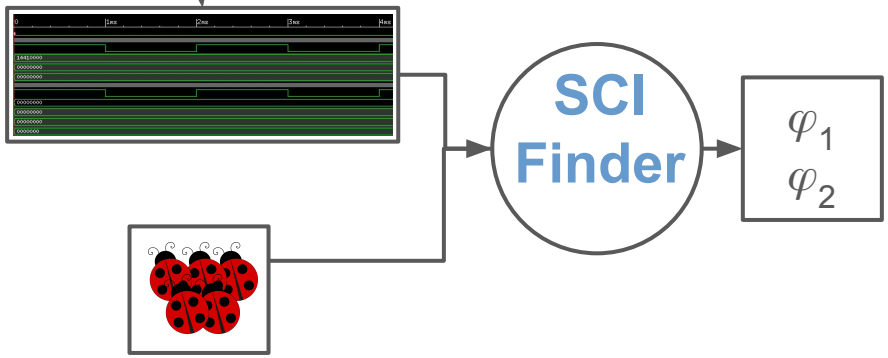

Traces
Miner
[Daikon]
Properties
Processor
Errata
Bug
Classification
[Manual]
Security
Bugs
Security
Property
Identification
Initial
Security
Properties
Logistic
Regression
φ
Security
Properties

Evaluation

- How well does SCIFinder identify security properties?
- Will the generated properties find security vulnerabilities?

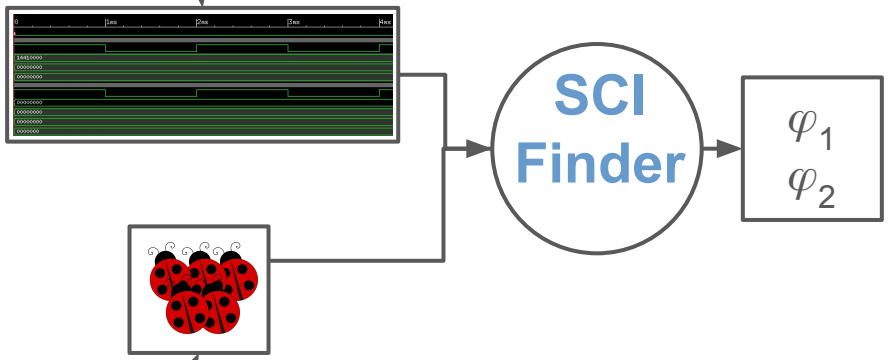
OR1200

- 26GB trace data
- 17 programs
- full instruction coverage



OR1200

- 26GB trace data
- 17 programs
- full instruction coverage

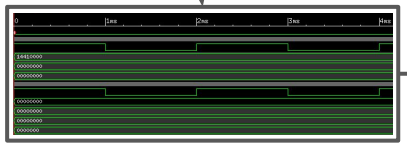


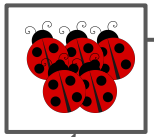
17 bugs

- 3 RISC architectures
- processor core

OR1200

- 26GB trace data
- 17 programs
- full instruction coverage





SCI Finder

φ_1
φ_2

17 bugs

- 3 RISC architectures
- processor core

87 properties

- 54 bug driven
- 33 model driven

Properties

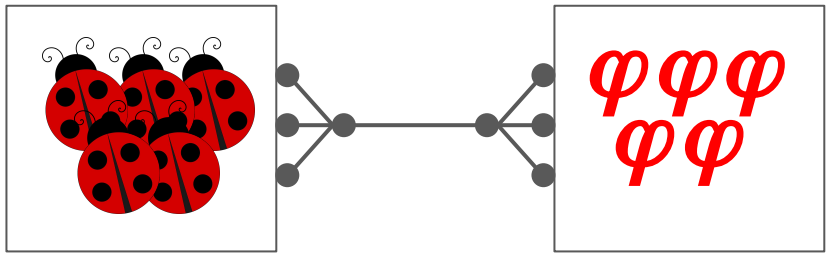
Bug Driven

- 54 properties from 17 bugs
- 47% false discovery rate

Properties

Bug Driven

- 54 properties from 17 bugs
- 47% false discovery rate



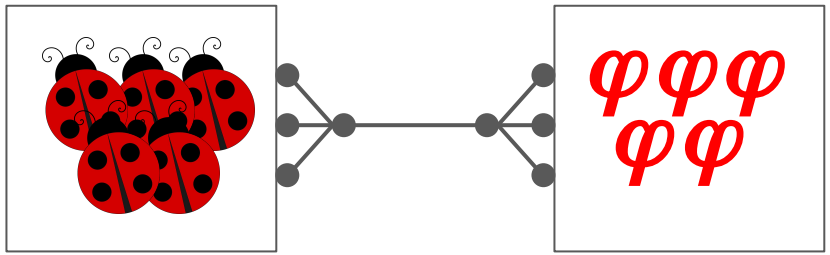
Properties

Bug Driven

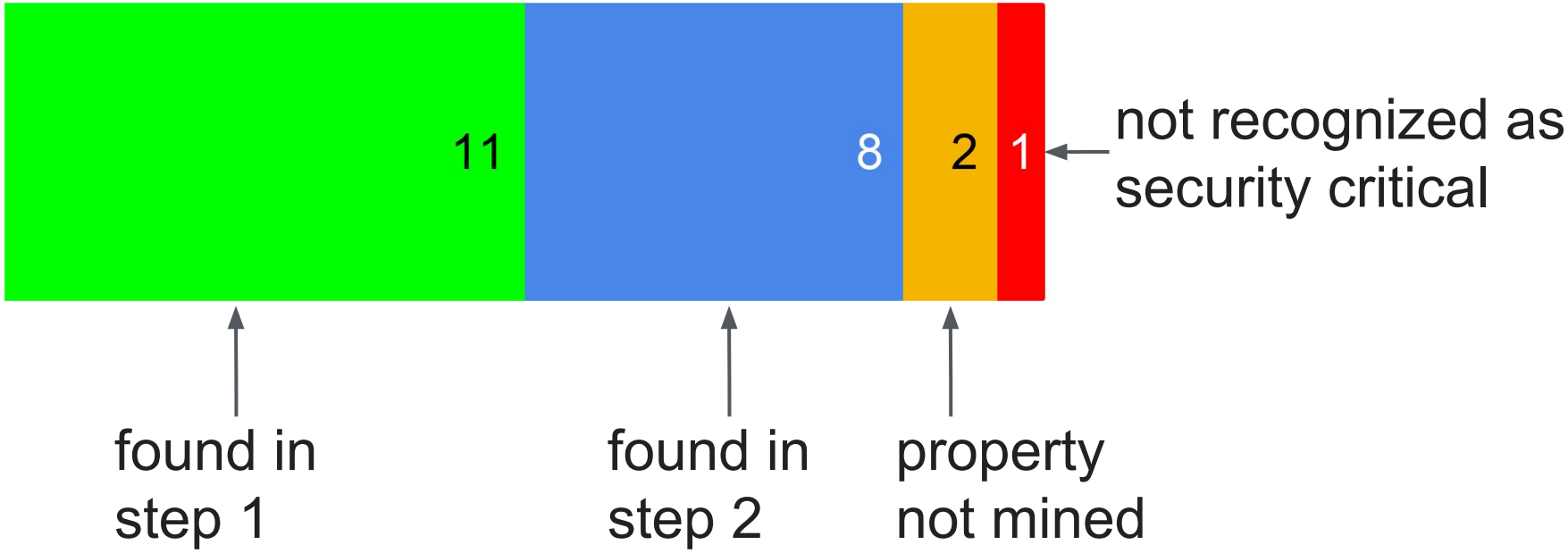
- 54 properties from 17 bugs
- 47% false discovery rate

Model Driven

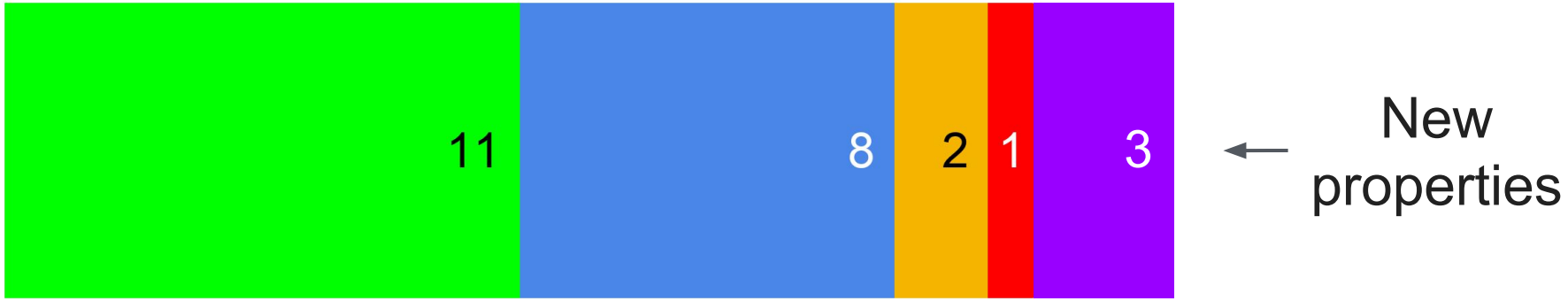
- 33 additional properties
- 27% false discovery rate



Comparison to State of the Art



Comparison to State of the Art



Finding and Exploiting Property Violations

[FMS 2018, MICRO 2018]

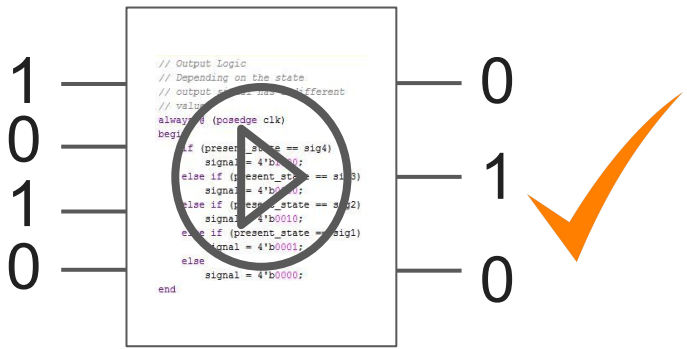
Problem Statement

Given φ and a processor design

- Can we find a violation of φ ?
- How do we reach the violating state?
- Can the violating state be exploited?

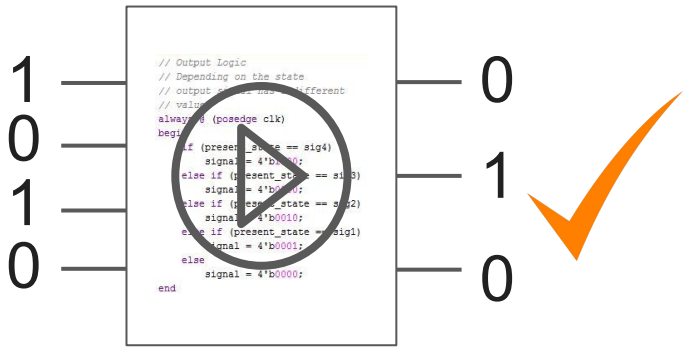
Existing Tools

Simulation Based Testing

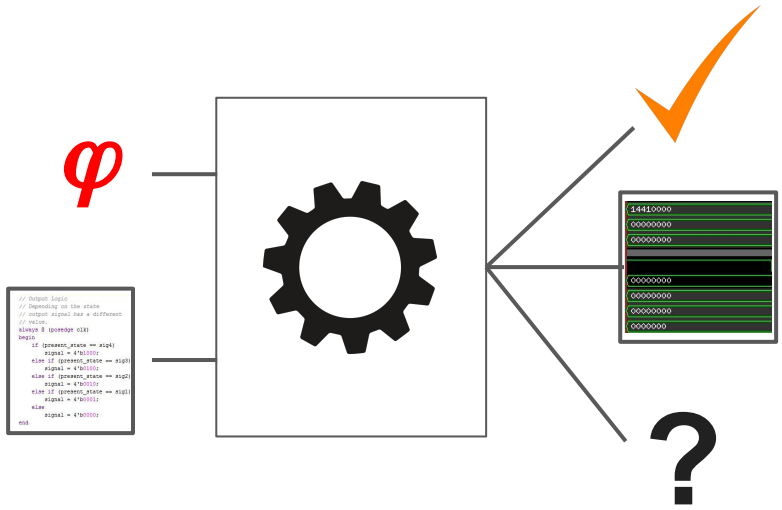


Existing Tools

Simulation Based Testing



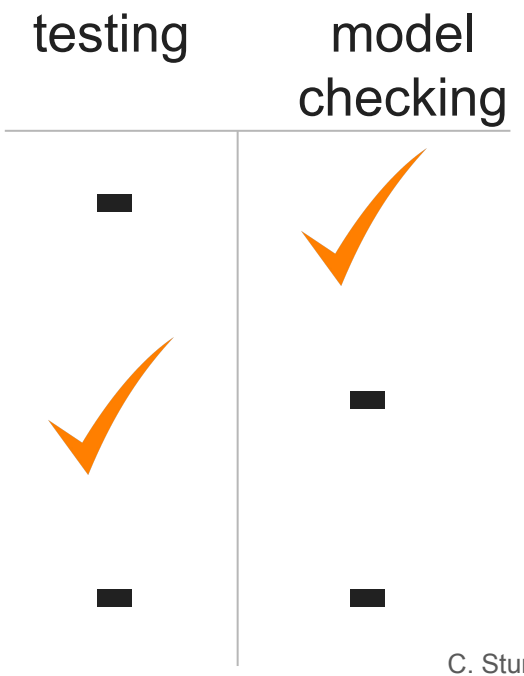
Model Checking



Problem Statement

Given φ and a processor design

- Can we find a violation of φ ?
- How do we reach the violating state?
- Can the violating state be exploited?



Symbolic Execution

```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```

	symbolic state
	reset := r_0 count := c_0

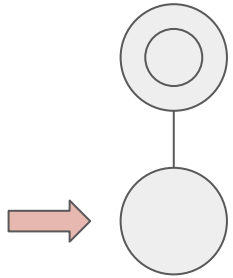
Symbolic Execution

```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```

path condition	symbolic state
True	reset := r_0 count := c_0

Symbolic Execution

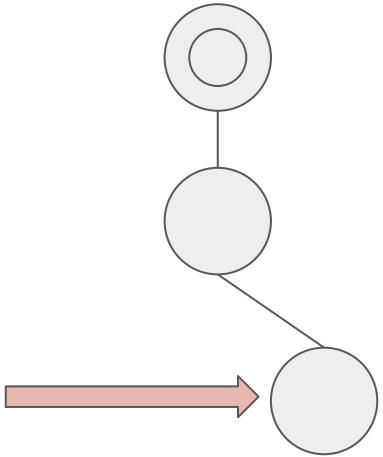
```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```



path condition	symbolic state
True	reset := r_0 count := c_0

Symbolic Execution

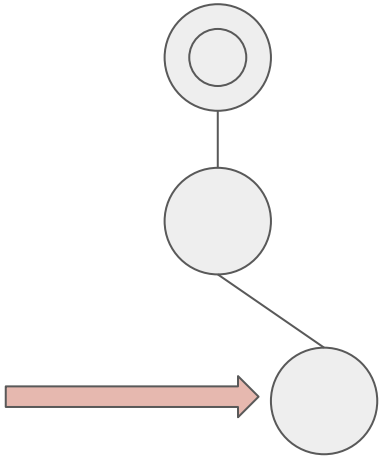
```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```



path condition	symbolic state
True	reset := r_0 count := c_0
$r_0 = 0$	

Symbolic Execution

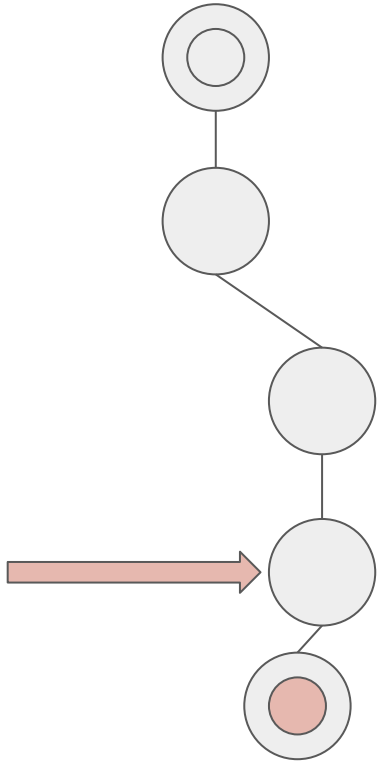
```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```



path condition	symbolic state
True	reset := r_0 count := c_0
$r_0 = 0$	reset := r_0 count := $c_0 + 1$

Symbolic Execution

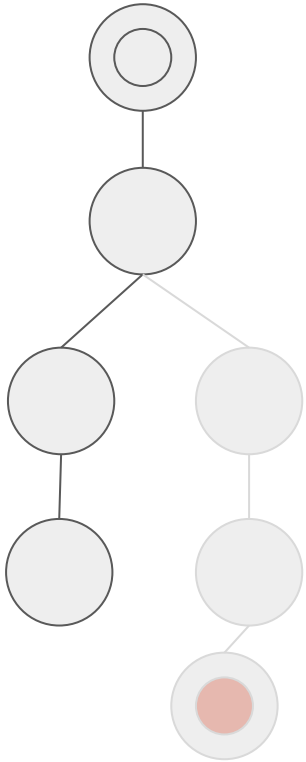
```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```



path condition	symbolic state
True	reset := r_0 count := c_0
$r_0 = 0$	reset := r_0 count := $c_0 + 1$
$r_0 = 0$ $c_0+1 > 3$	

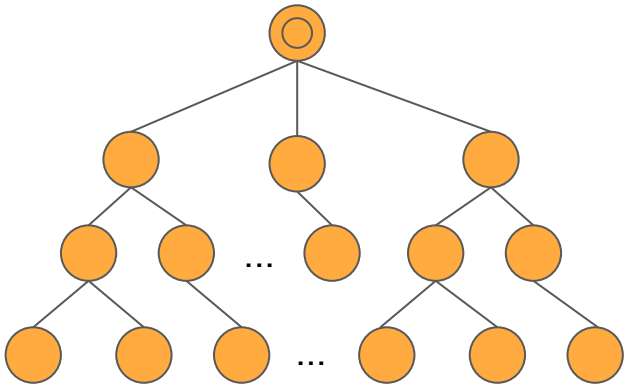
Symbolic Execution

```
if (reset)
  count = 0;
else
  count = count+1;

if (count > 3)
  ERROR;
```

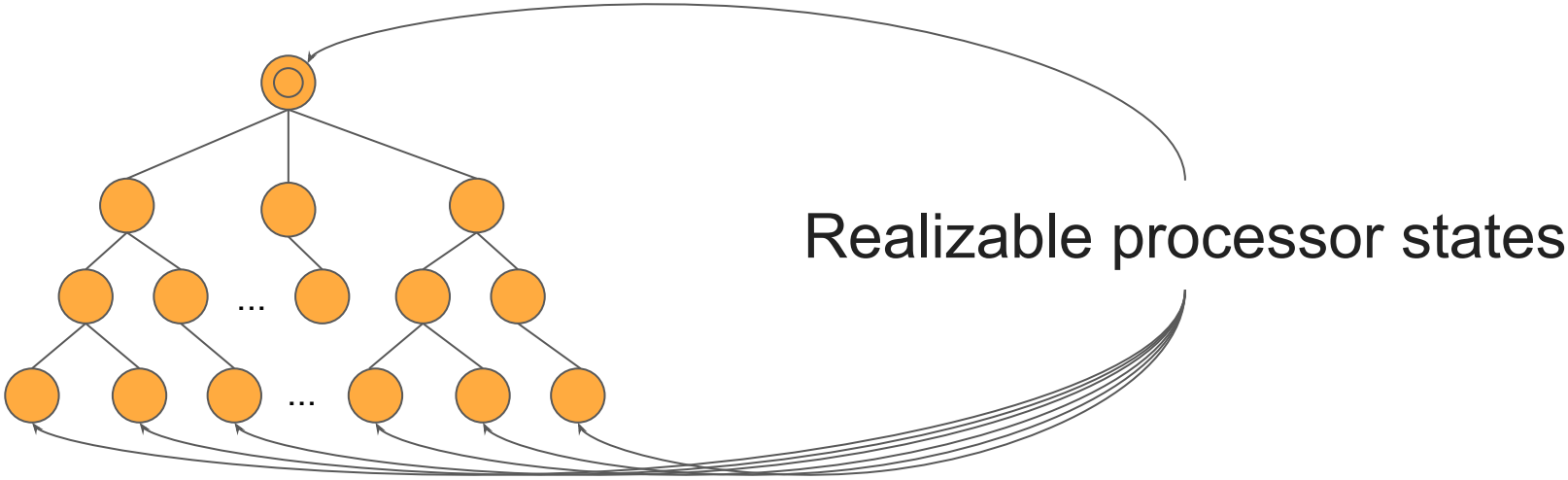


path condition	symbolic state
True	reset := r_0 count := c_0
$r_0 = 1$	reset := r_0 count := 0
$r_0 = 1$ $0 > 3$	

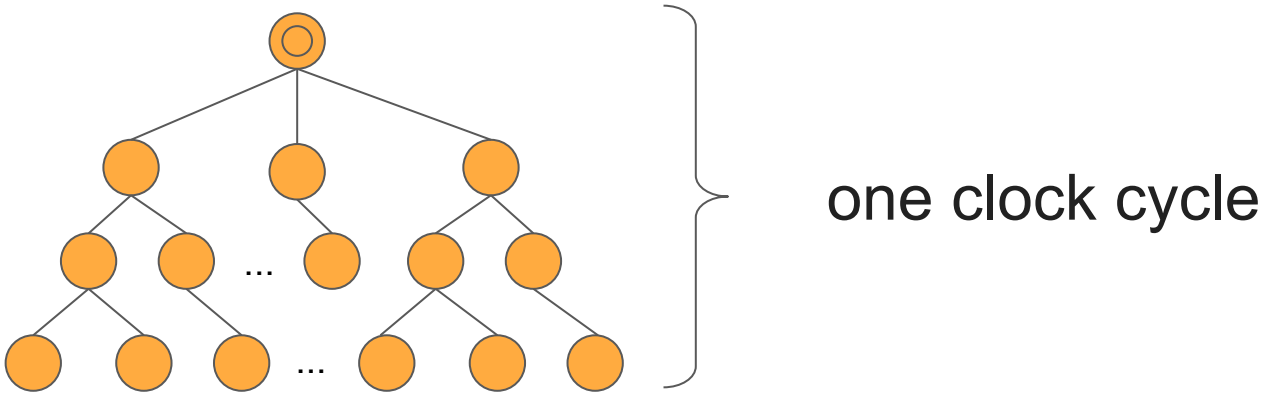
Symbolic Execution of a Hardware Design



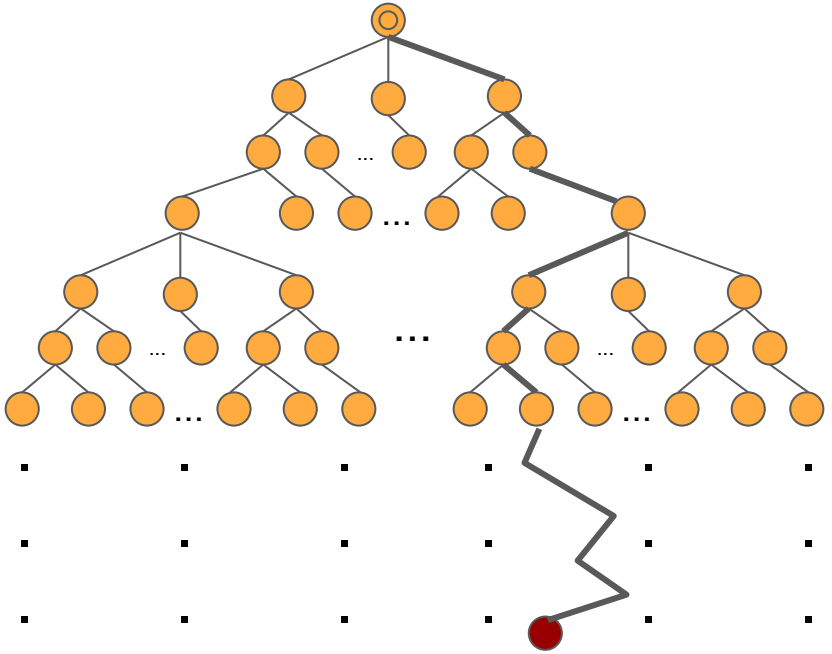
Symbolic Execution of a Hardware Design



Symbolic Execution of a Hardware Design

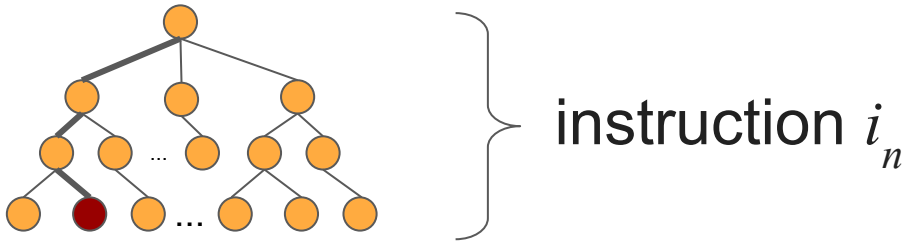


Symbolic Execution of a Hardware Design

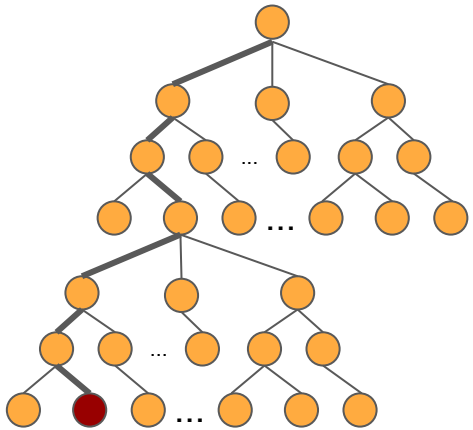


one clock cycle

Backward Search

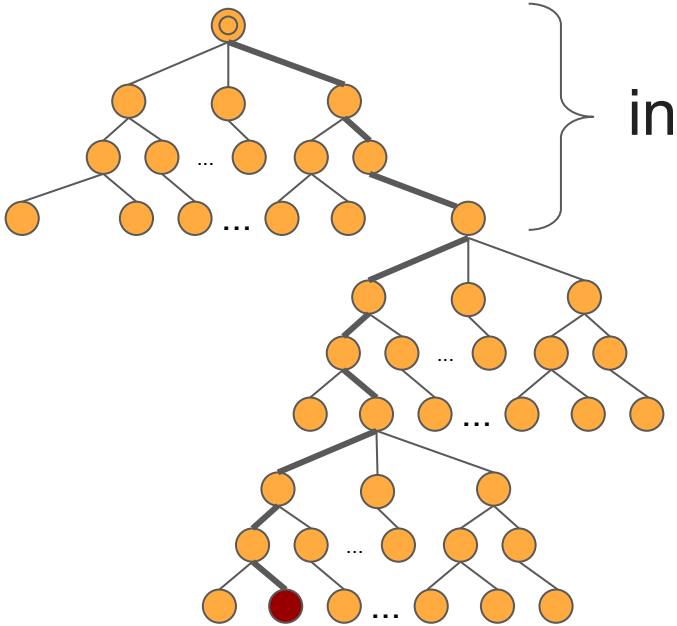


Backward Search



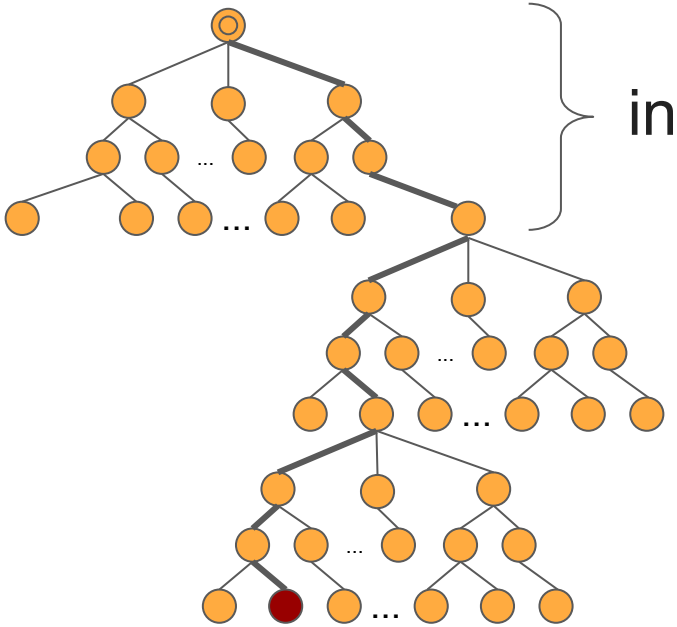
instruction i_{n-1}

Backward Search



instruction i_{n-2}

Backward Search



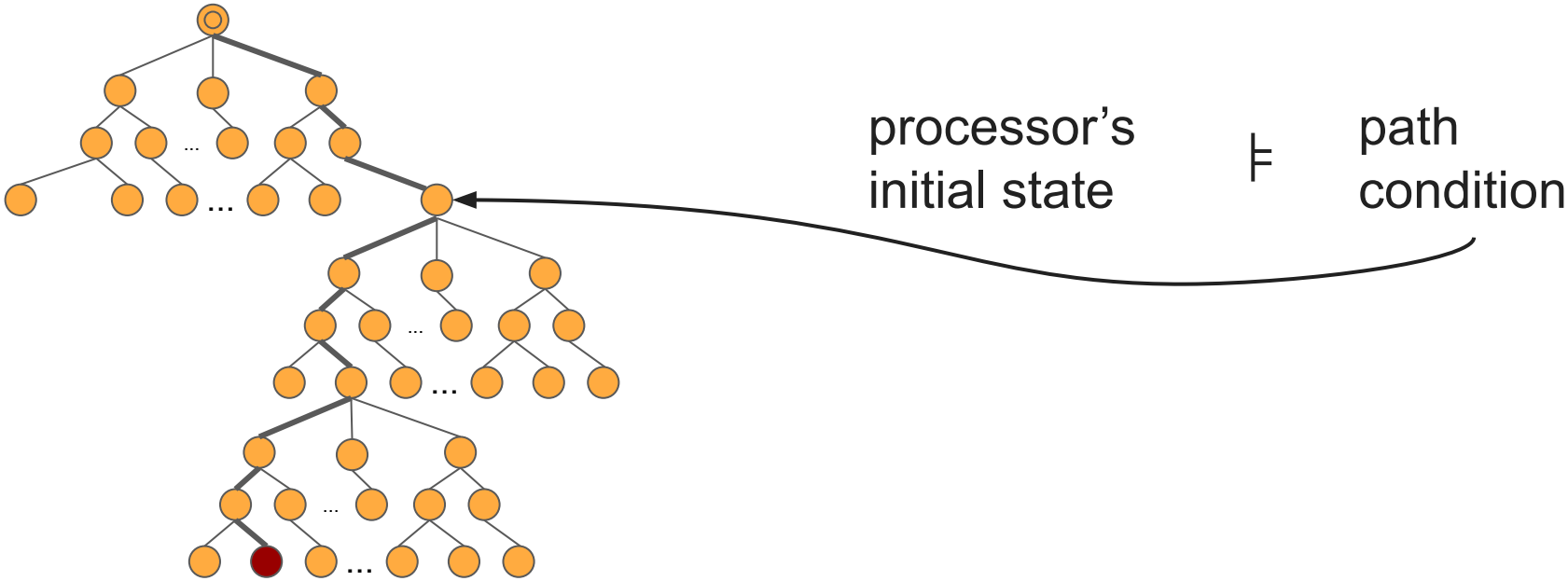
instruction i_{n-2}

trigger:

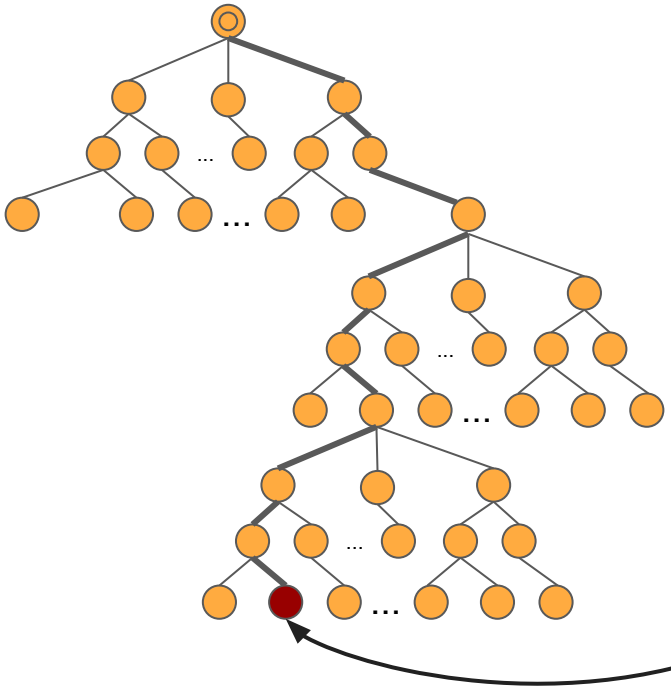
$i_0, i_1, \ldots, i_{n-2}, i_{n-1}, i_n$

If a sequence of inputs is returned, it will take the processor from the initial state to an error state.

Requirements

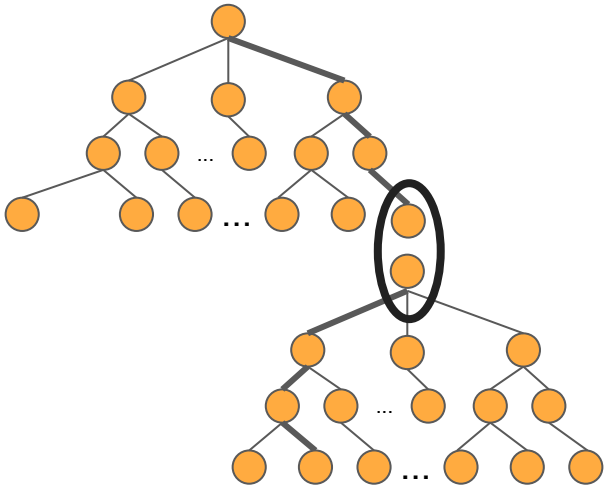


Requirements

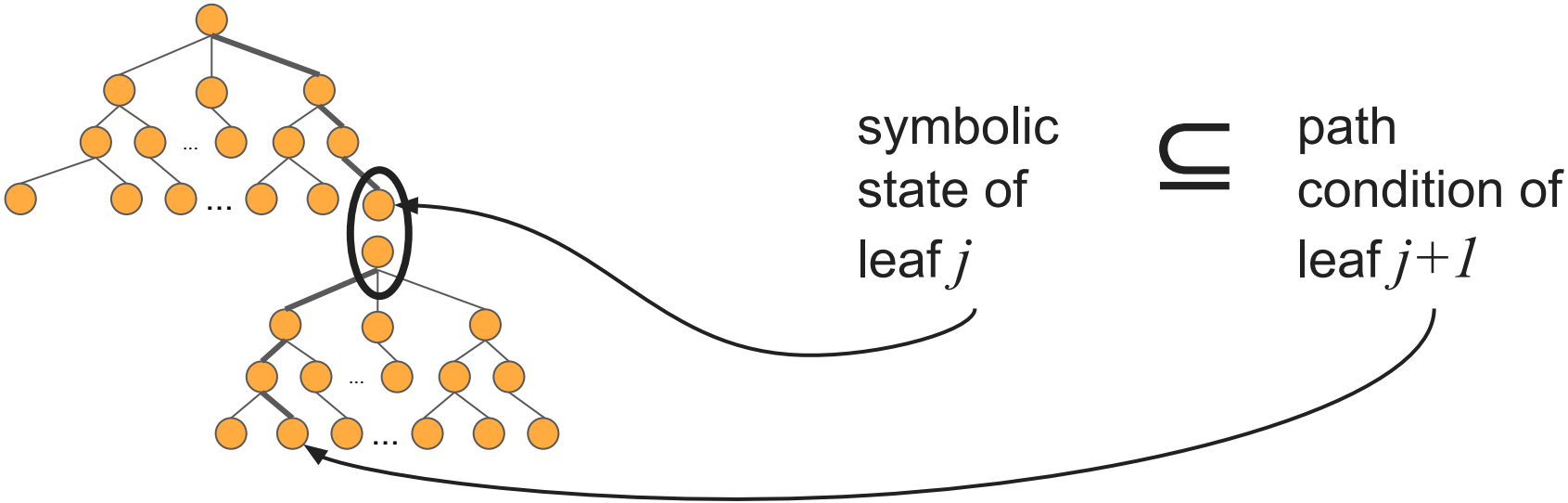


symbolic state $\subseteq$ assertion failure

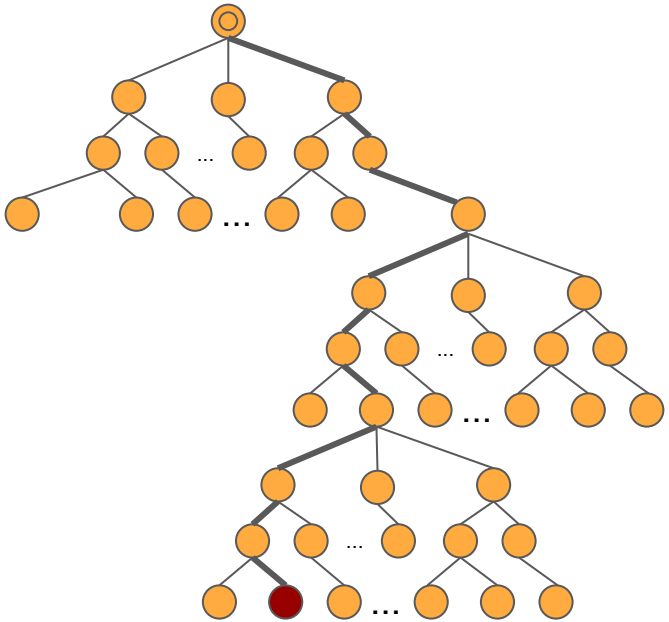
Requirements



Requirements

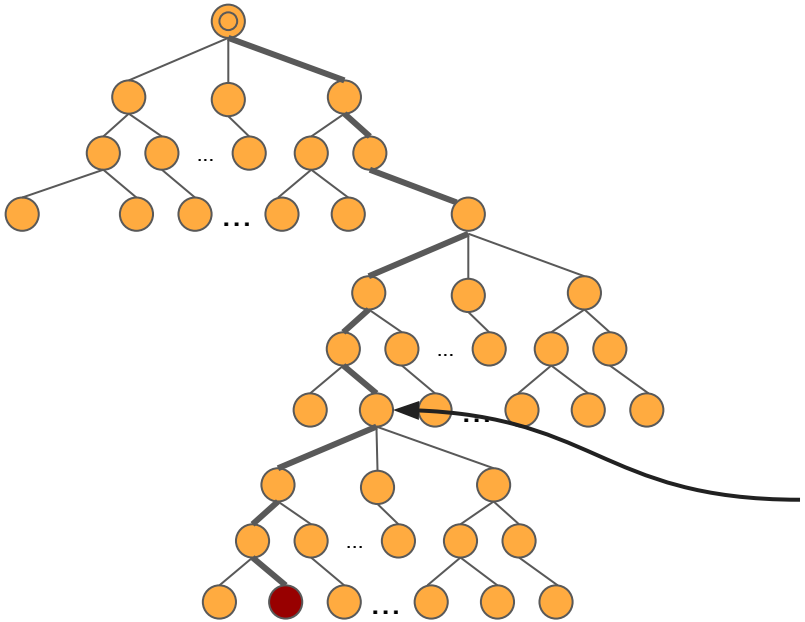


Making it Work



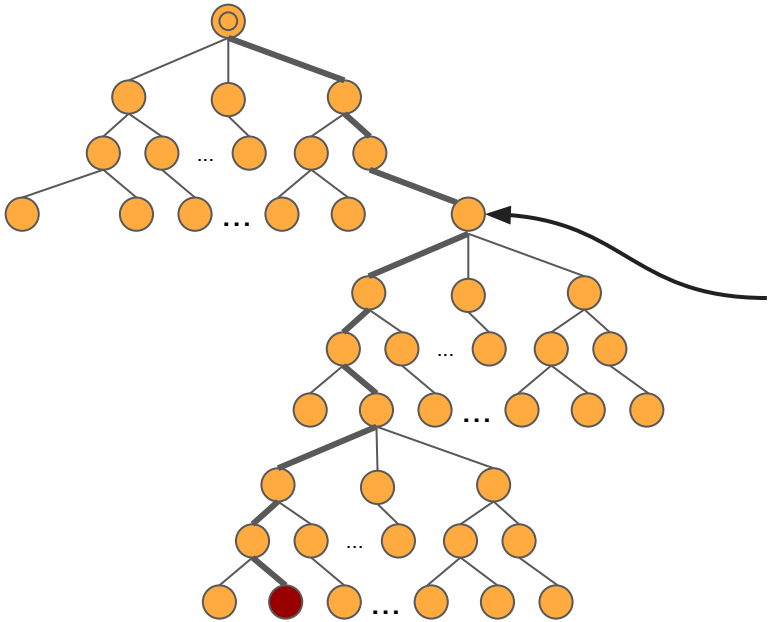
1. Internal and input signals are symbolic

Making it Work



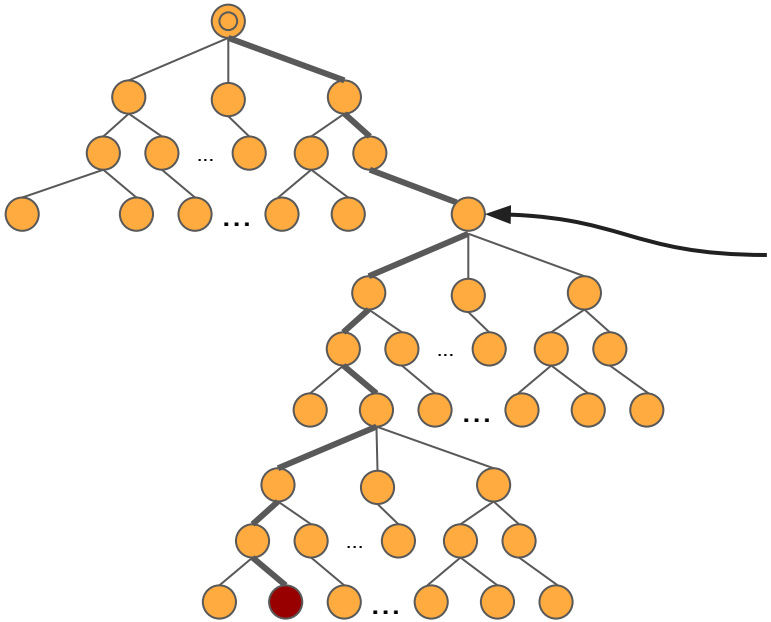
2. Is this an initial state?

Making it Work



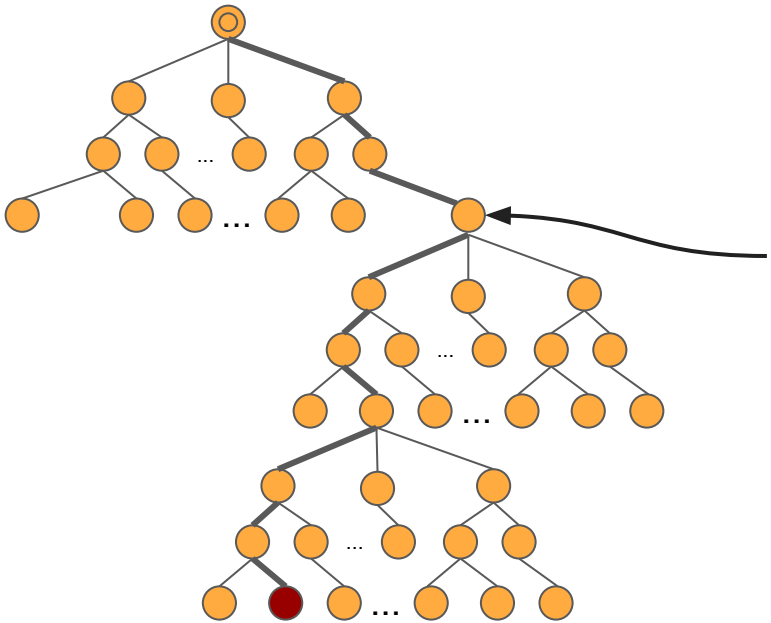
3. How much does this differ from an initial state?

Making it Work

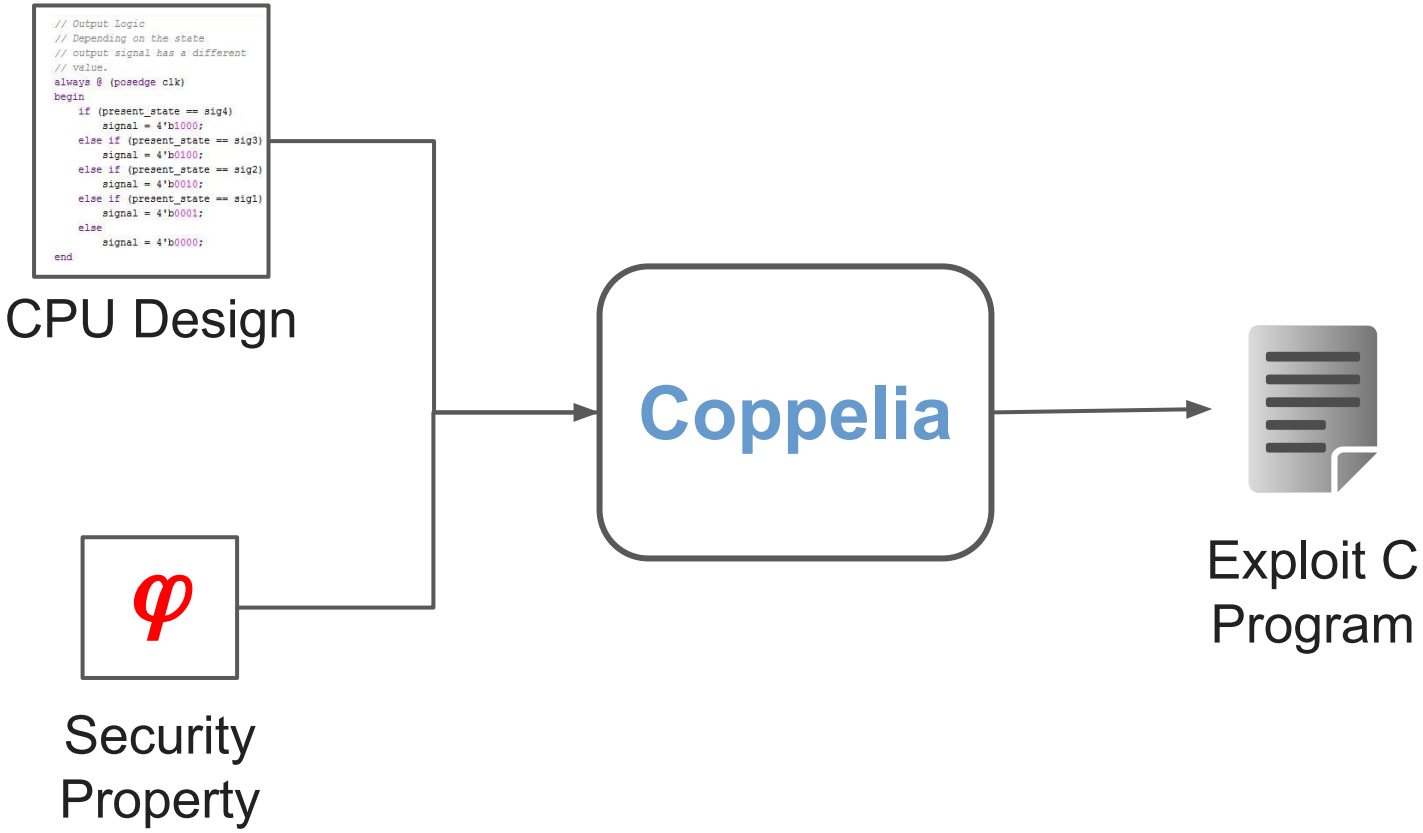


3. Are we in a loop?

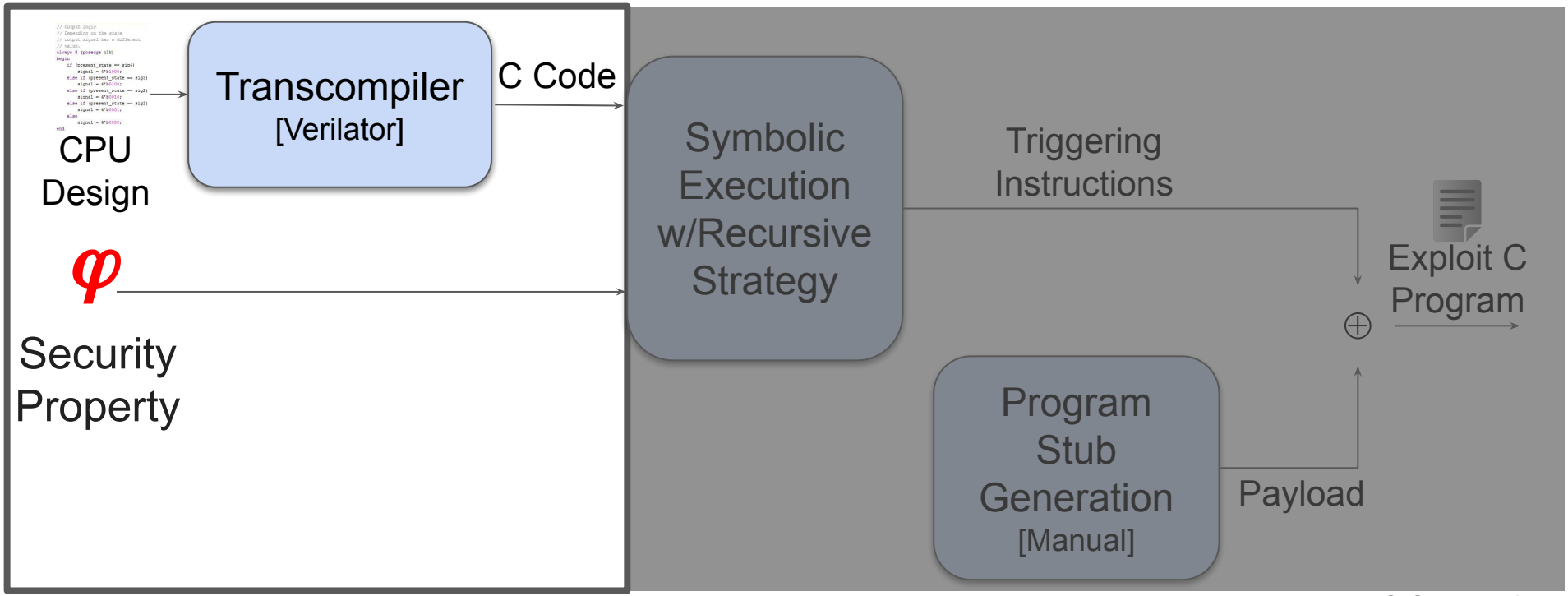
Making it Work



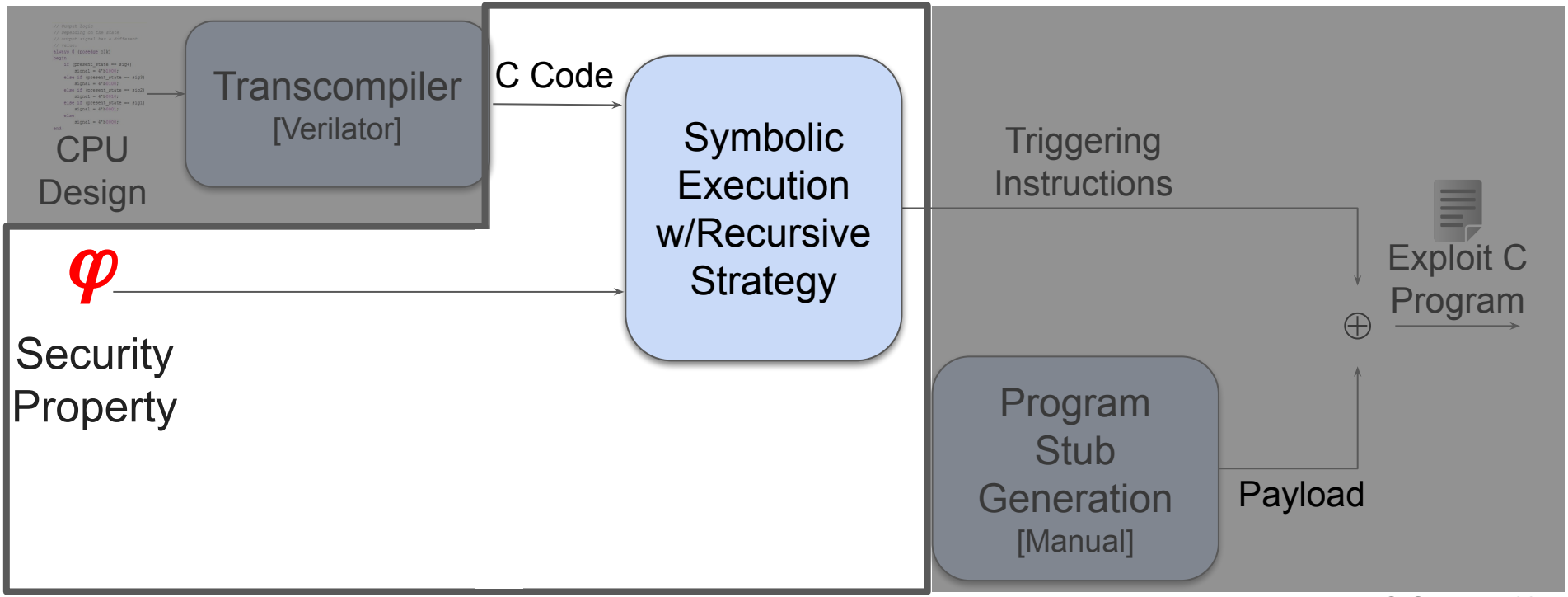
4. Have we exceeded the bound?


// Output Logic
// Depending on the state
// output signal has a different
// value.
always @ (posedge clk)
begin
if (present_state == sig4)
signal = 4'b1000;
else if (present_state == sig3)
signal = 4'b0100;
else if (present_state == sig2)
signal = 4'b0010;
else if (present_state == sig1)
signal = 4'b0001;
else
signal = 4'b0000;
end
CPU Design
φ
Security Property
Coppelia
Exploit C Program

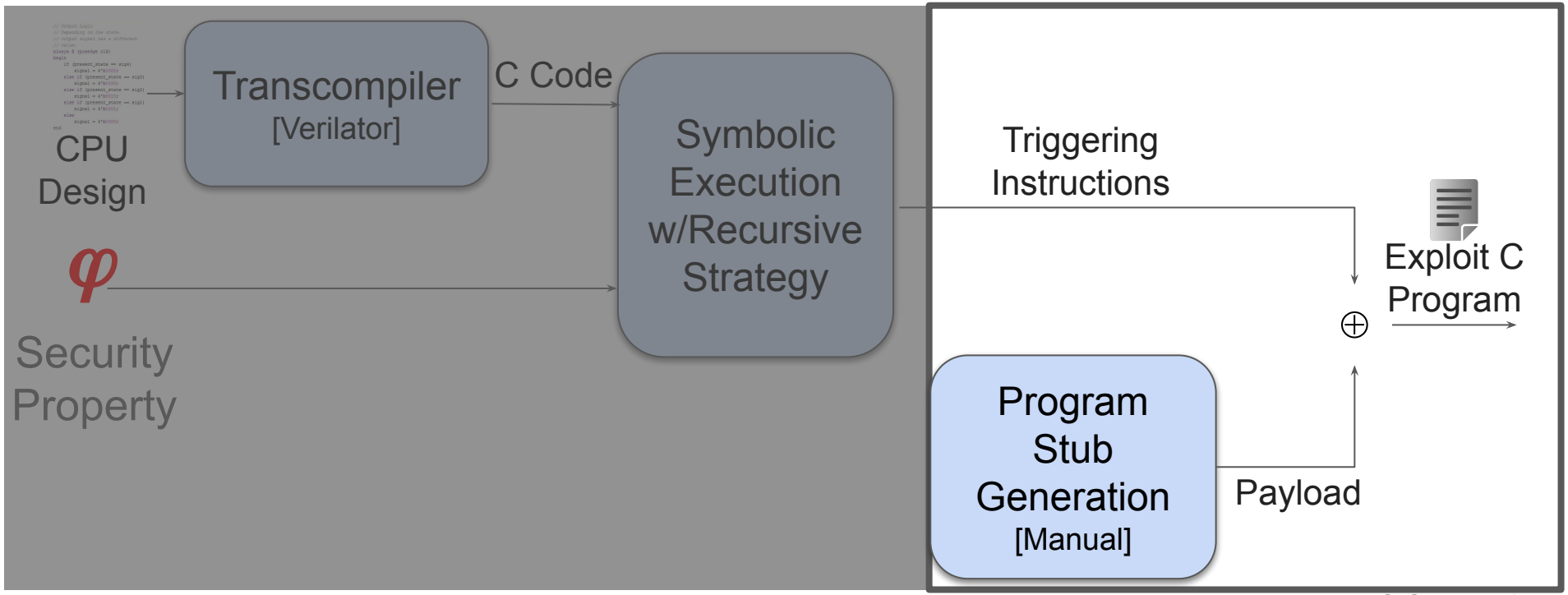
Coppelia


CPU Design
Transcompiler
[Verilator]
C Code
φ
Security Property
Symbolic Execution w/Recursive Strategy
Triggering Instructions
Exploit C Program
Program Stub Generation
[Manual]
Payload

Coppelia


CPU Design
Transcompiler
[Verilator]
C Code
Symbolic Execution w/Recursive Strategy
Security Property
Triggering Instructions
Exploit C Program
Program Stub Generation
[Manual]
Payload

Coppelia


CPU Design
Transcompiler
[Verilator]
C Code
Symbolic Execution w/Recursive Strategy
Security Property
Triggering Instructions
Exploit C Program
Program Stub Generation
[Manual]
Payload

Optimizations

- Explore only legal instructions
- Alternate depth-first and breadth-first searching
- Cone of Influence analysis for slicing

Evaluating Optimizations

Baseline	DFS + BFS	Cone of Influence
> 19h	> 1.2h	4m 12s

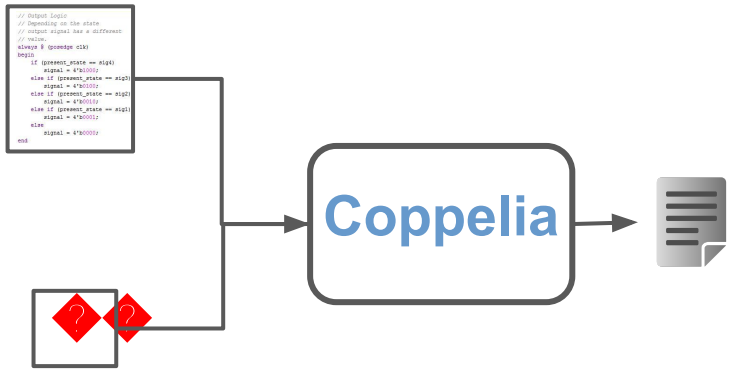
- Average CPU time to find bug
- Considered only bugs triggerable with a single instruction

Evaluating Coppelia

- Does Coppelia find bugs and generate their exploits?
- Will our approach find new bugs?

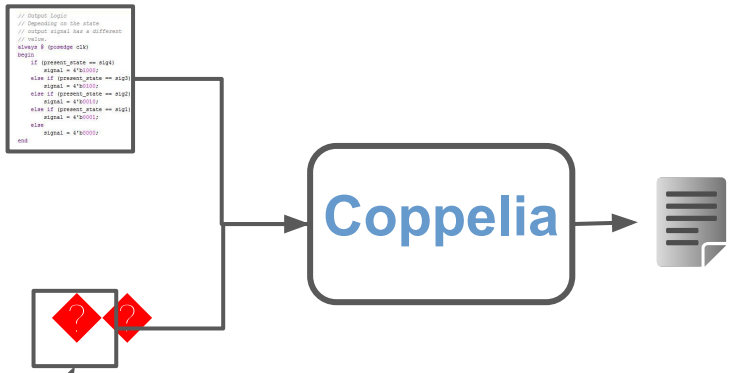
Processor

- OR1200
- 31 known bugs



Processor

- OR1200
- 31 known bugs

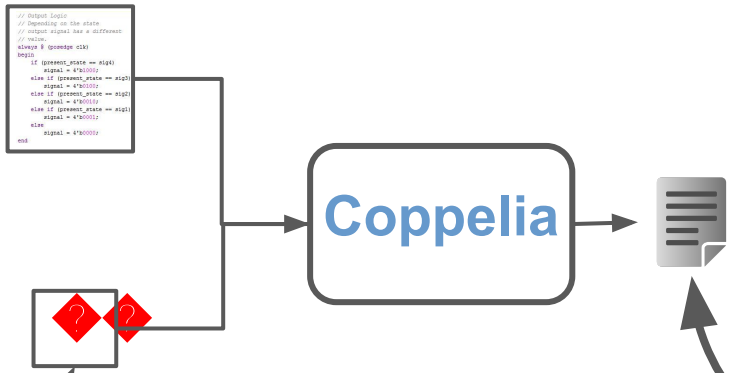


35 properties

- SPECS
- SecurityCheckers
- SCIFinder

Processor

- OR1200
- 31 known bugs

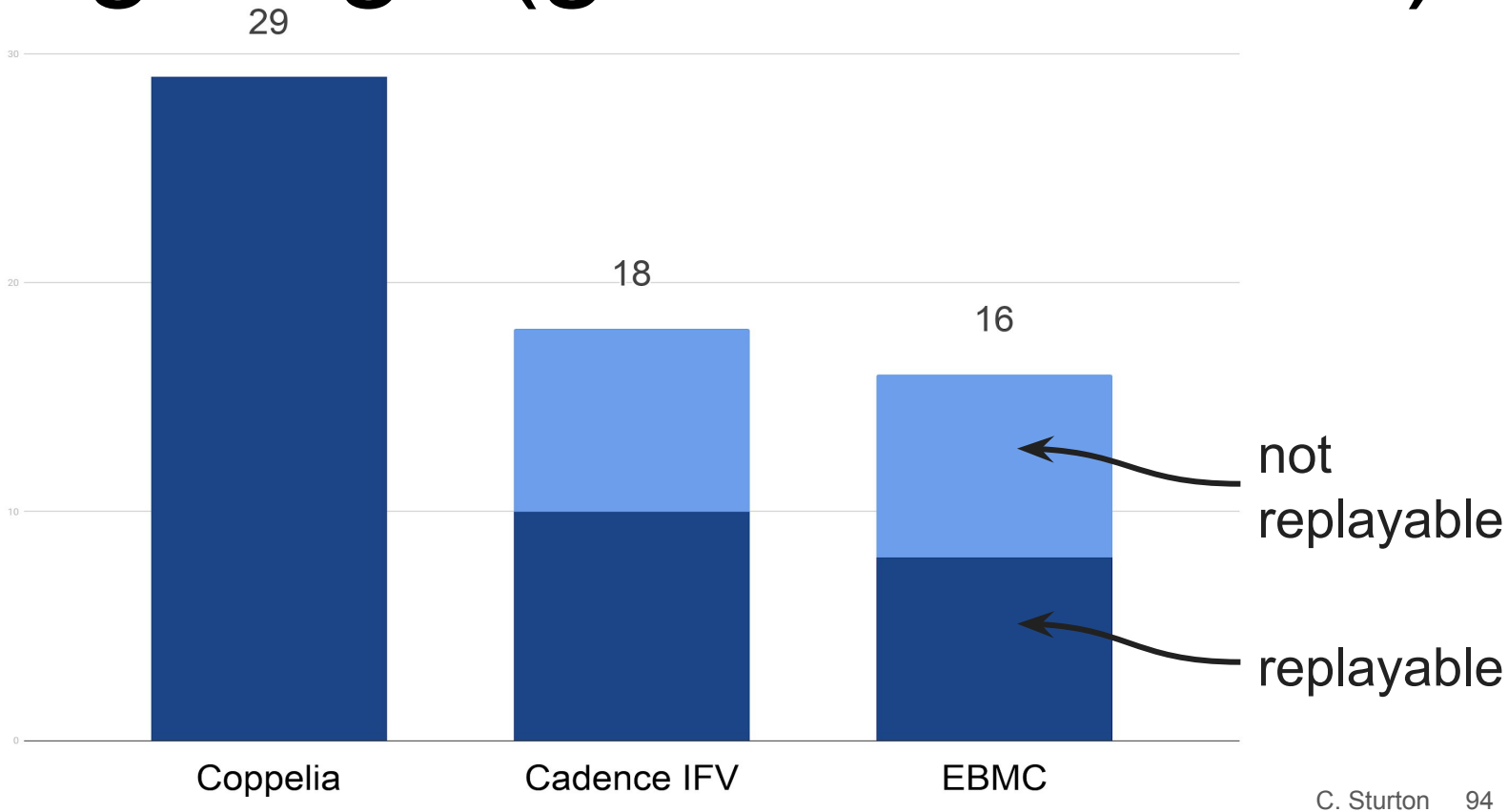


35 properties

- SPECS
- SecurityCheckers
- SCIFinder

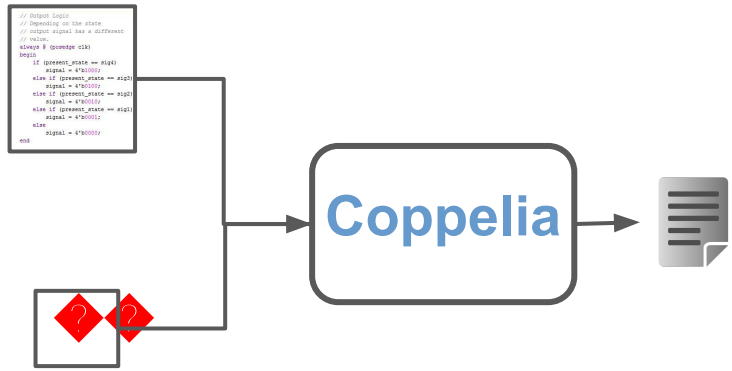
29 exploits

Finding Bugs (ground truth: 31)



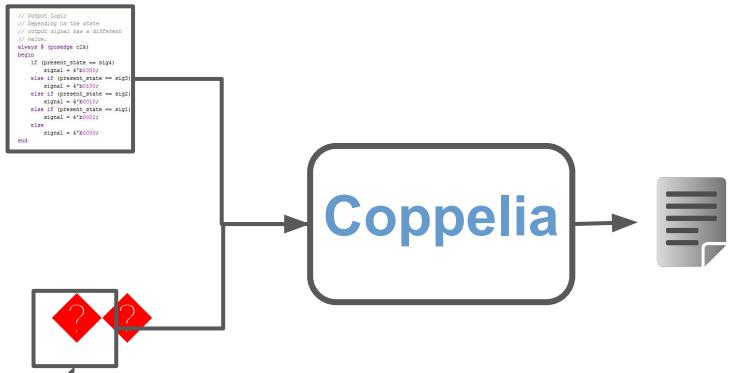
Processor

- Mor1kx
- PULPino



Processor

- Mor1kx
- PULPino

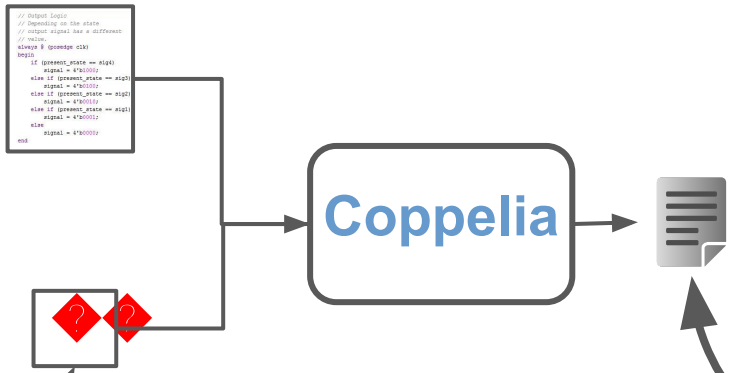


Properties

- SPECS
- SecurityCheckers
- SCIFinder

Processor

- Mor1kx
- PULPino



Properties

- SPECS
- SecurityCheckers
- SCIFinder

4 new bugs

Finding New Bugs

1

Mor1kx-Espresso

new
design

3

PULPino-RI5CY

new
architecture!

Security validation of hardware designs can be done algorithmically

Our Products So Far

- **SCIFinder** to produce security critical properties
- **Coppelia** to find and generate exploits for property violations
- **Security properties** for RISC processor designs

Thank you

Rui Zhang, Calvin Deutschbein, Natalie Stanley,
Chris Griggs, Andrew Chi, Ryan Huang, Alyssa Byrnes,
Matthew Hicks, Jonathan M. Smith, Sam T. King.

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Hardware Security @ UNC