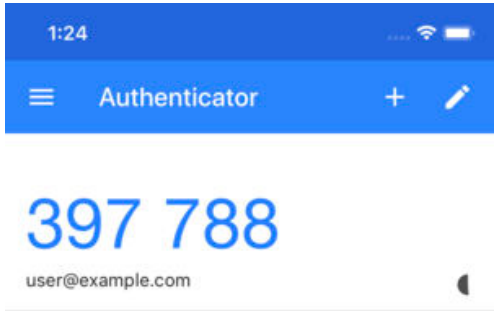
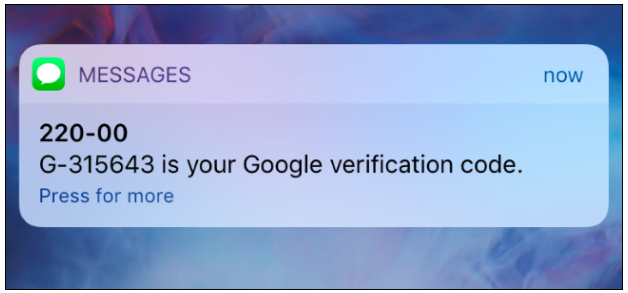
Want strong isolation? Just reset your processor.

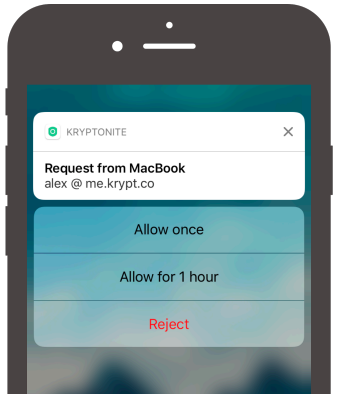
How we can build more secure systems by applying the age-old wisdom of “turning it off and on again”

Anish Athalye, Adam Belay, Frans Kaashoek, Robert Morris, Nickolai Zeldovich

Security devices are increasingly common







Smartphone apps



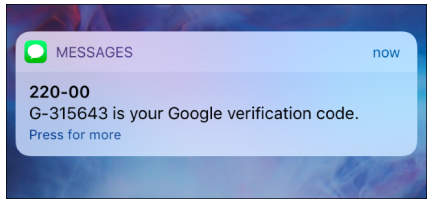




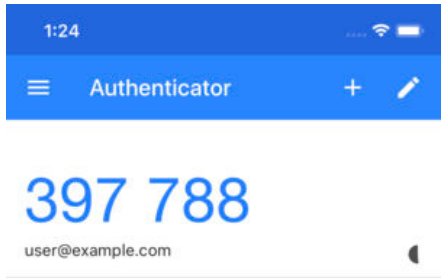
Custom hardware

They are getting better

SMS 2FA



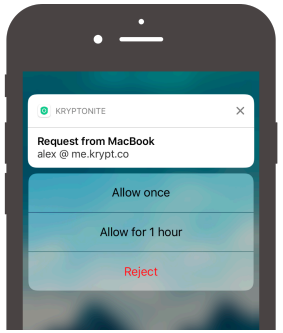
Smartphone TOTP



Hardware TOTP



Smartphone U2F



Hardware U2F

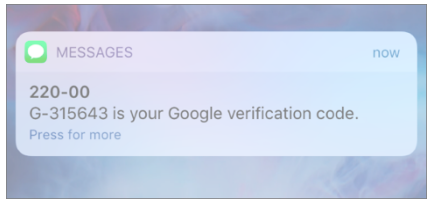


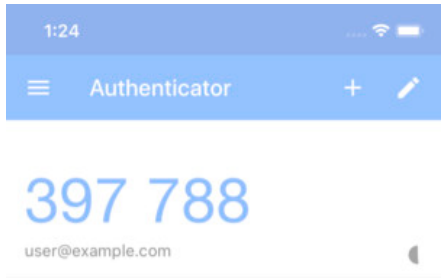
Hardware Cryptocurrency wallet



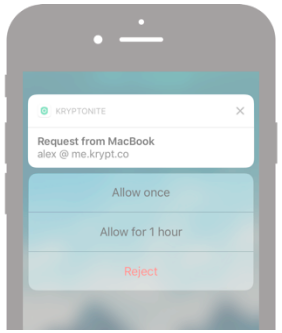
Smaller TCB

Paradigm shift











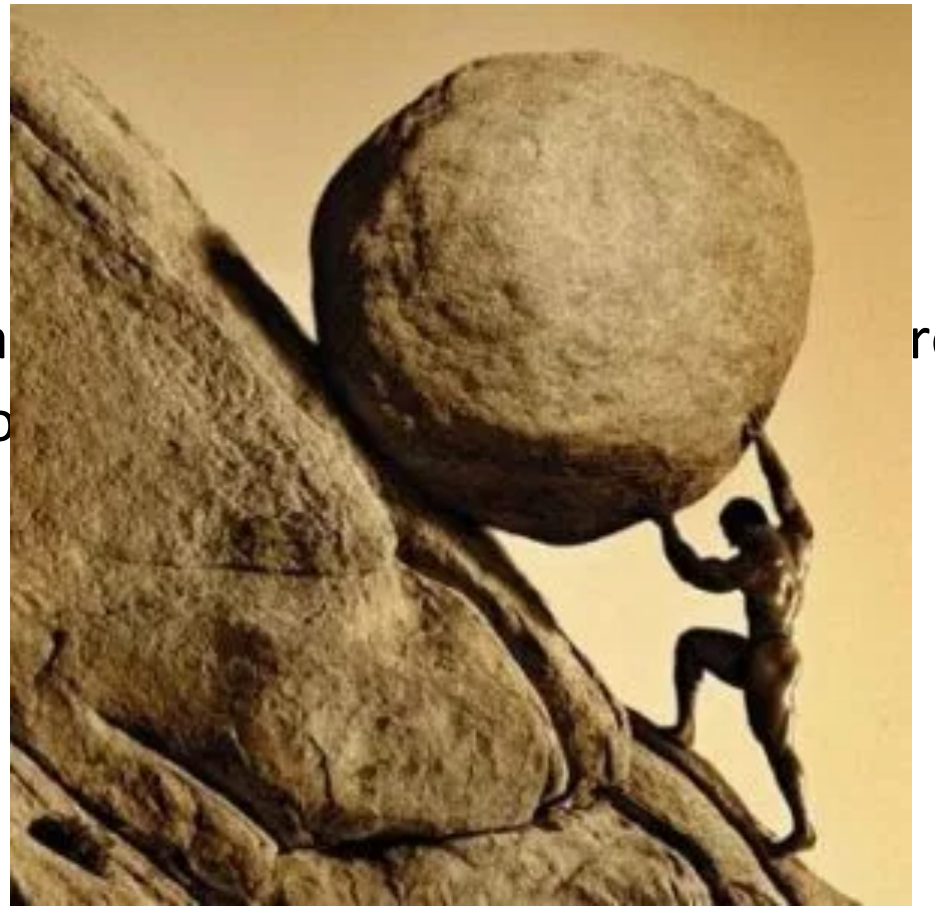


2FA: More secure login on PC

Transaction approval,
Removes PC from TCB

Can we make the PC secure instead?

- Endless bugs:
  - Application bugs
  - OS bugs (kernels >
  - Micro-architectura reshadow, Zombieload)
  - Hardware bugs (Ro



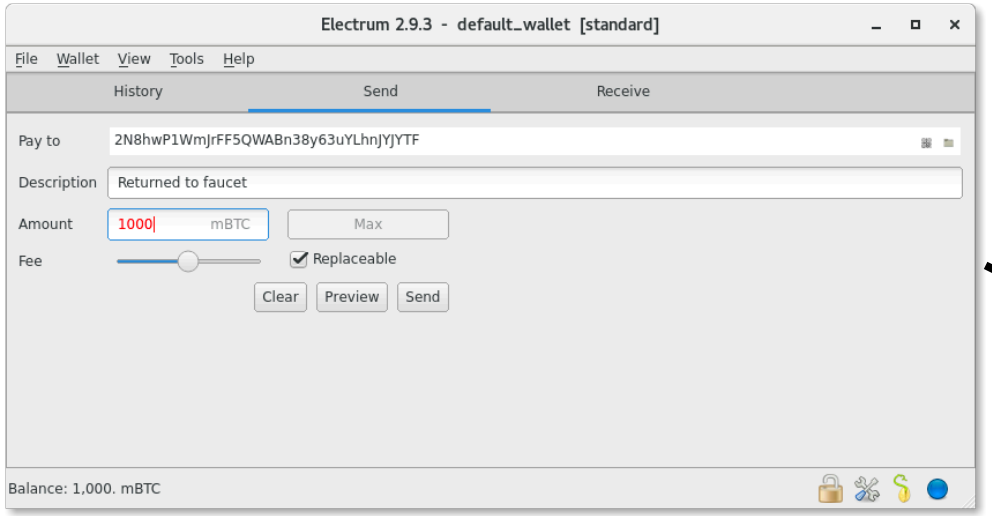
Transaction approval on simple devices



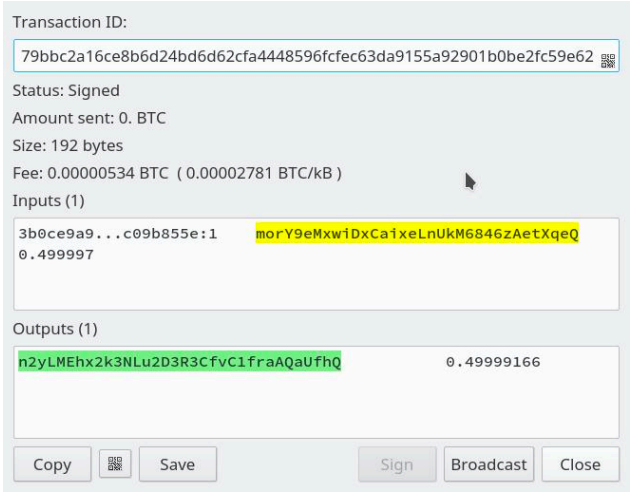




Remove PC from TCB



Untrusted PC



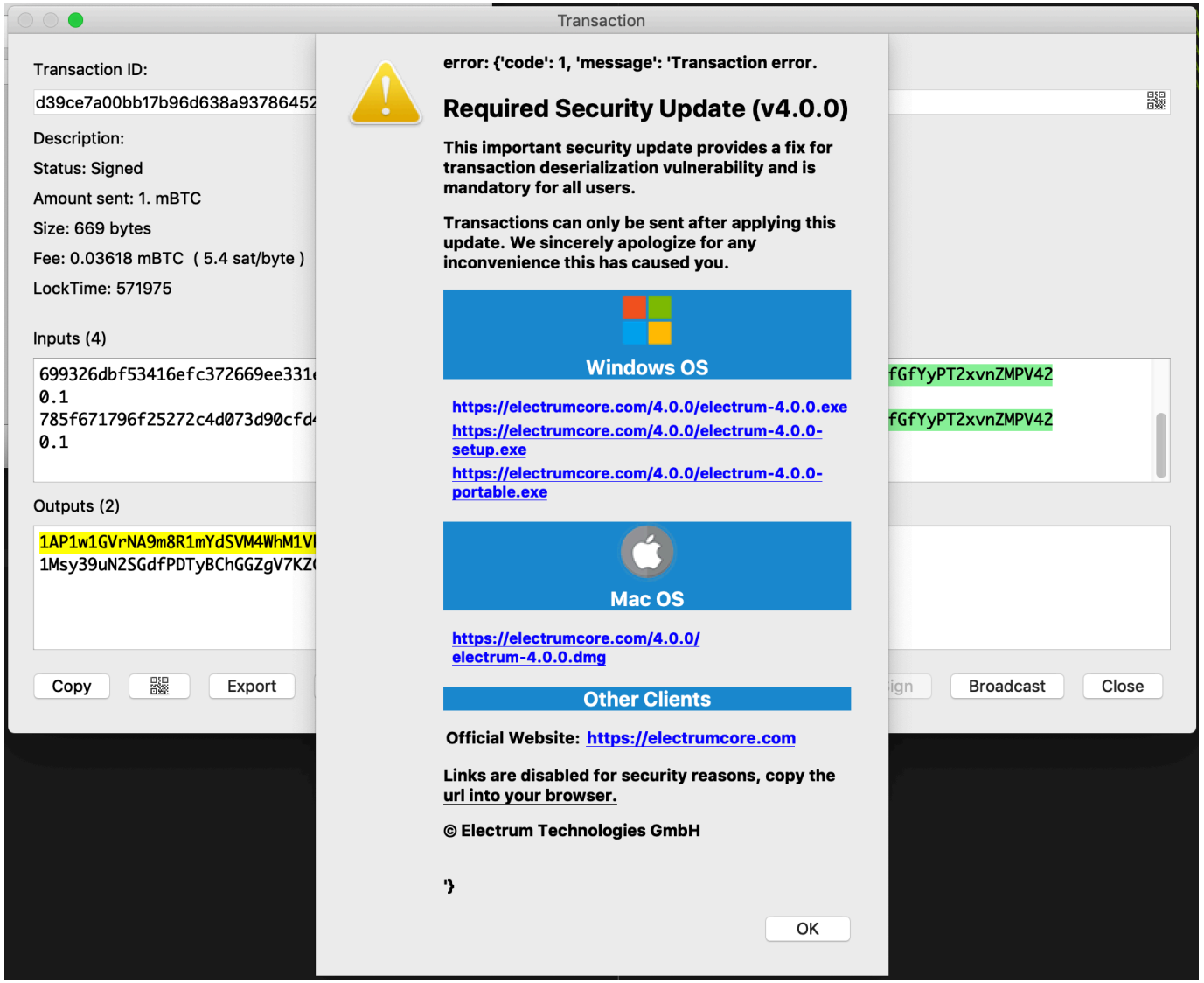
TX

Sign(TX)

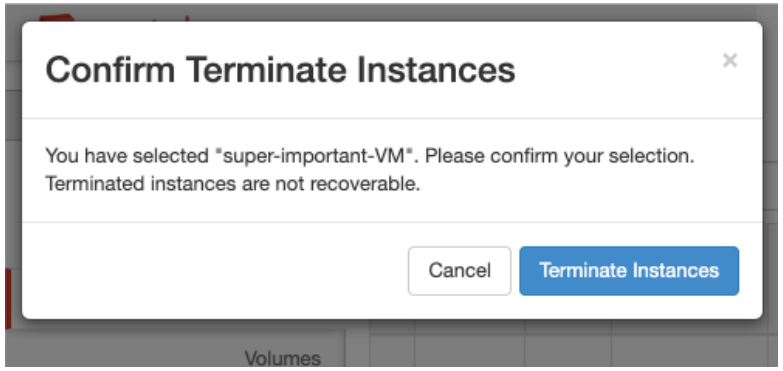


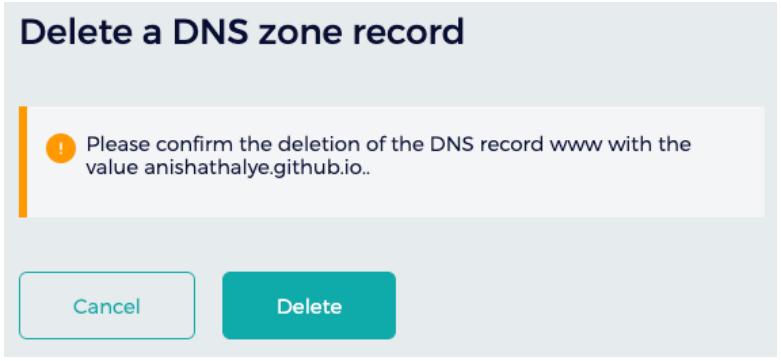
Trusted hardware
(has private key)

Remove PC from TCB

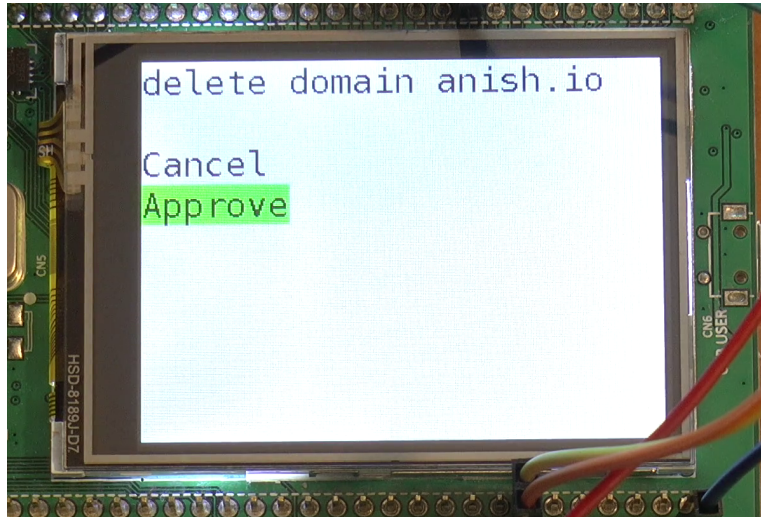


Not just cryptocurrencies





Before: confirm on PC



After: must confirm on device, which signs transaction

Not just cryptocurrencies

Web Authentication: An API for accessing Public Key Credentials Level 1

W3C Recommendation, 4 March 2019

§ 10.2. Simple Transaction Authorization Extension (txAuthSimple)

This extension allows for a simple form of transaction authorization. A Relying Party can specify a prompt string, intended for display on a trusted device on the authenticator.

Transaction approval everywhere!



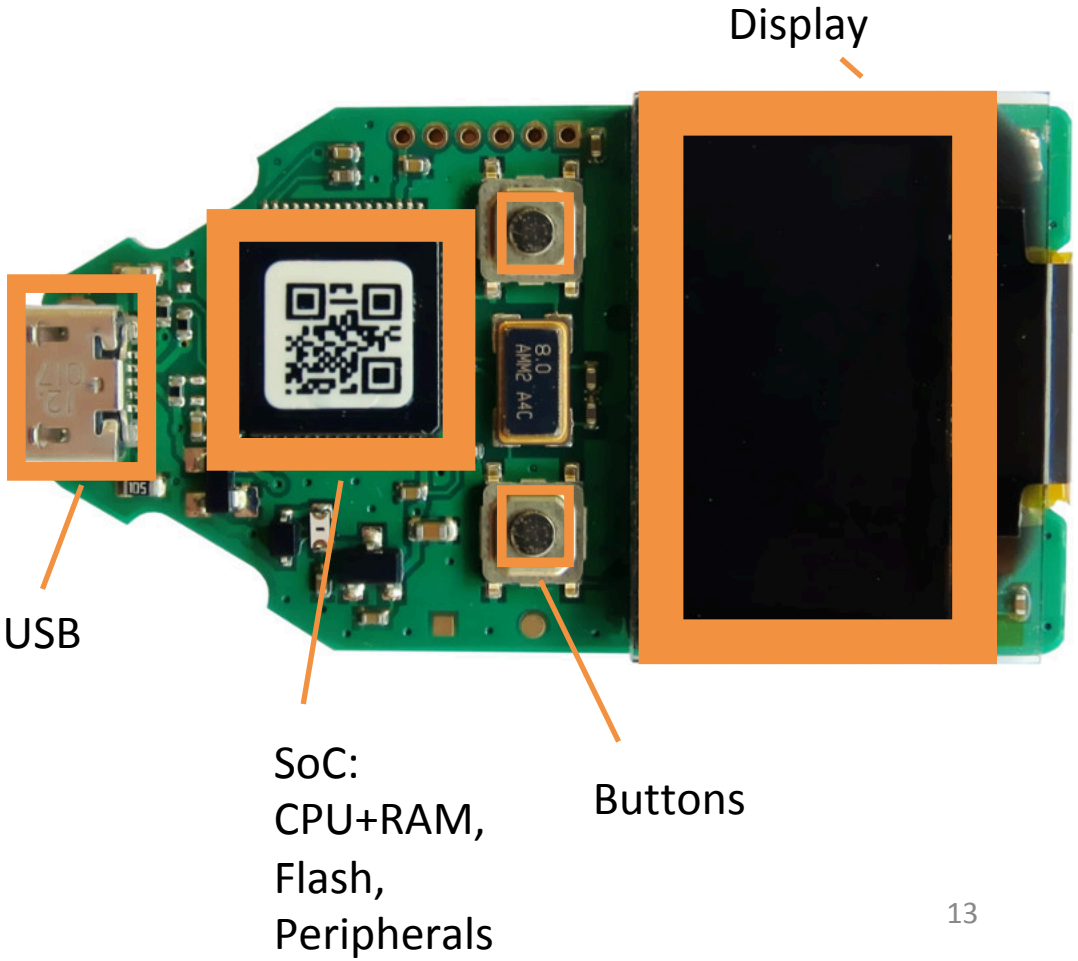
Transaction approval everywhere!



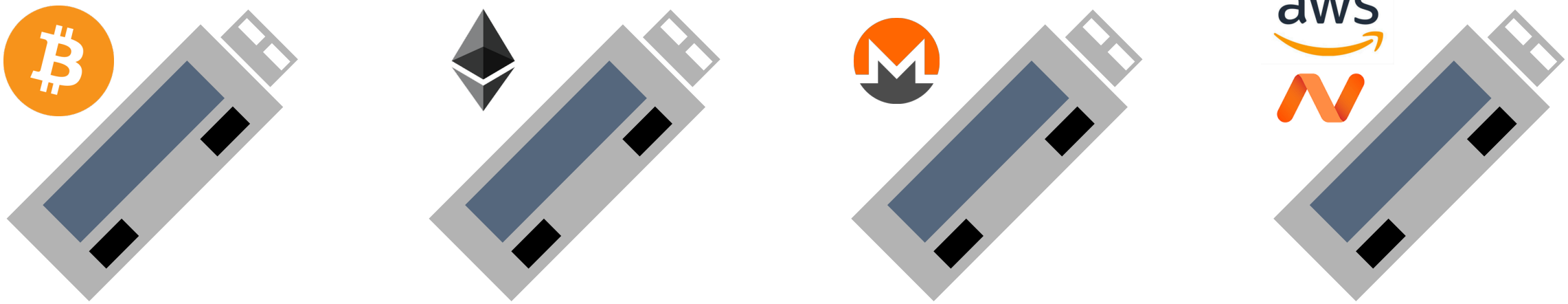
Sharing results in isolation bugs

- Some past wallet bugs
  - Bad argument validation in syscalls
  - Bad configuration of MPU

[Riscure @ Black Hat 2018; Ledger Blog; Trezor Blog]



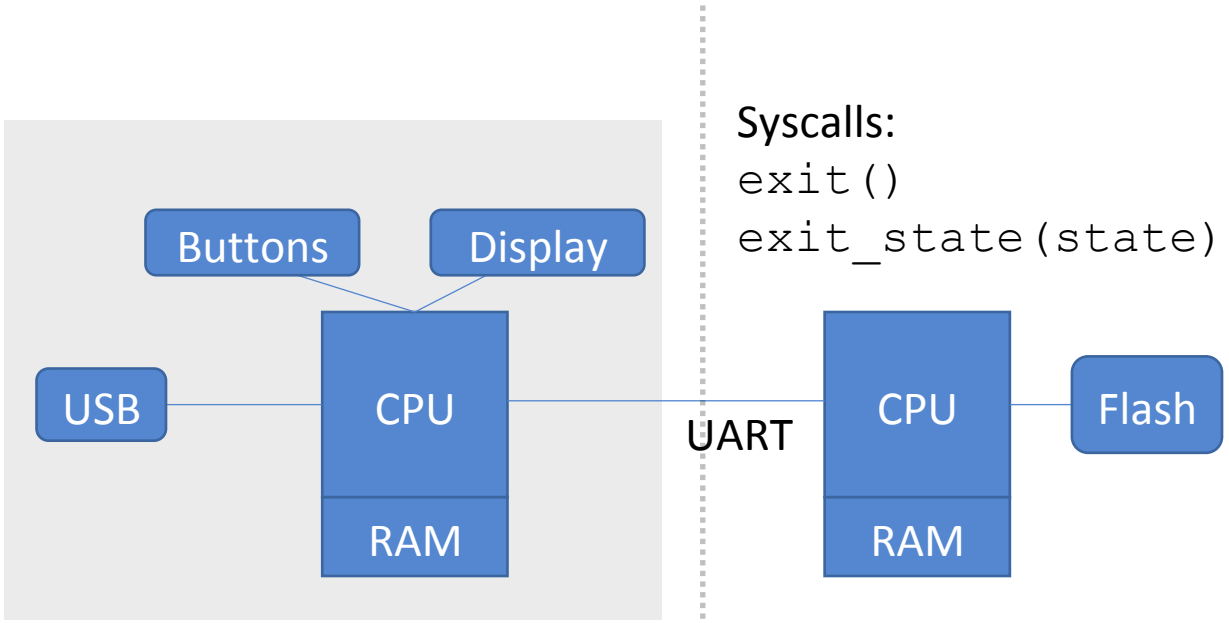
Security through physical separation



Simulating physical separation



Reset-based design



Syscalls:

```
exit()
exit_state(state)
```

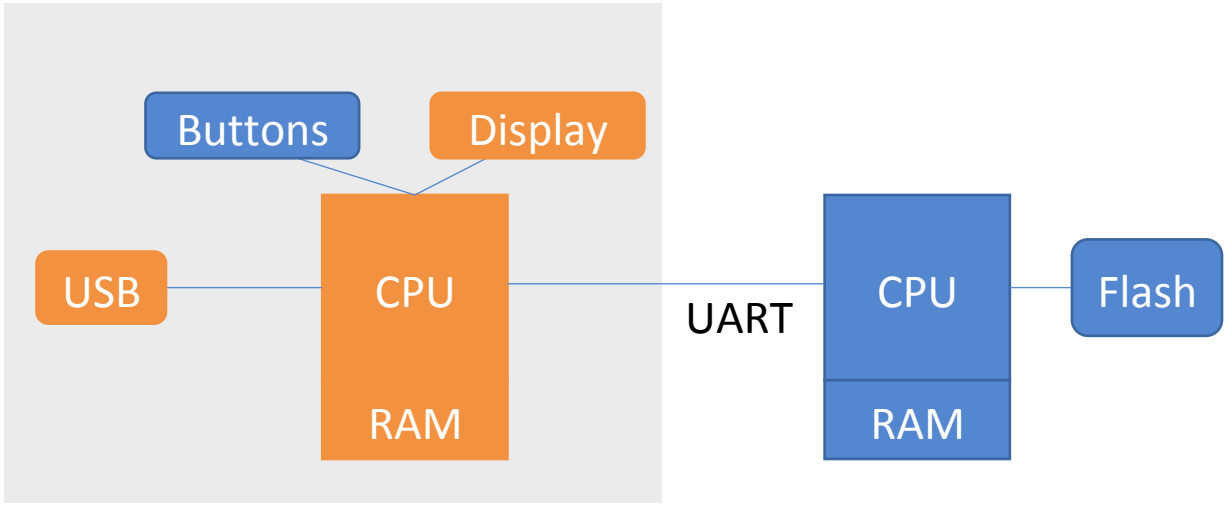
Application core

Runs third-party code;
has no persistent state

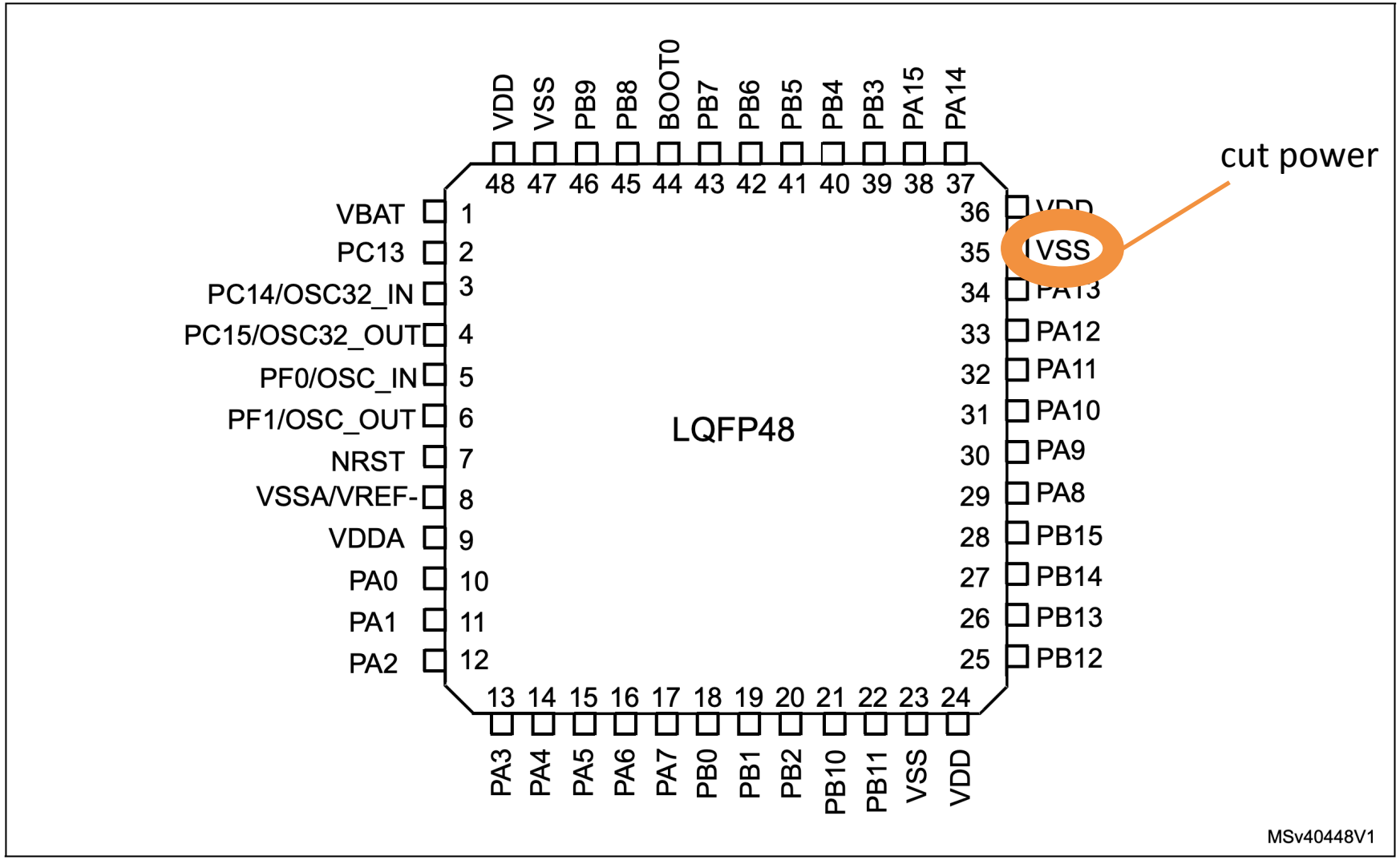
Management core

Manages persistent state;
never runs third-party code

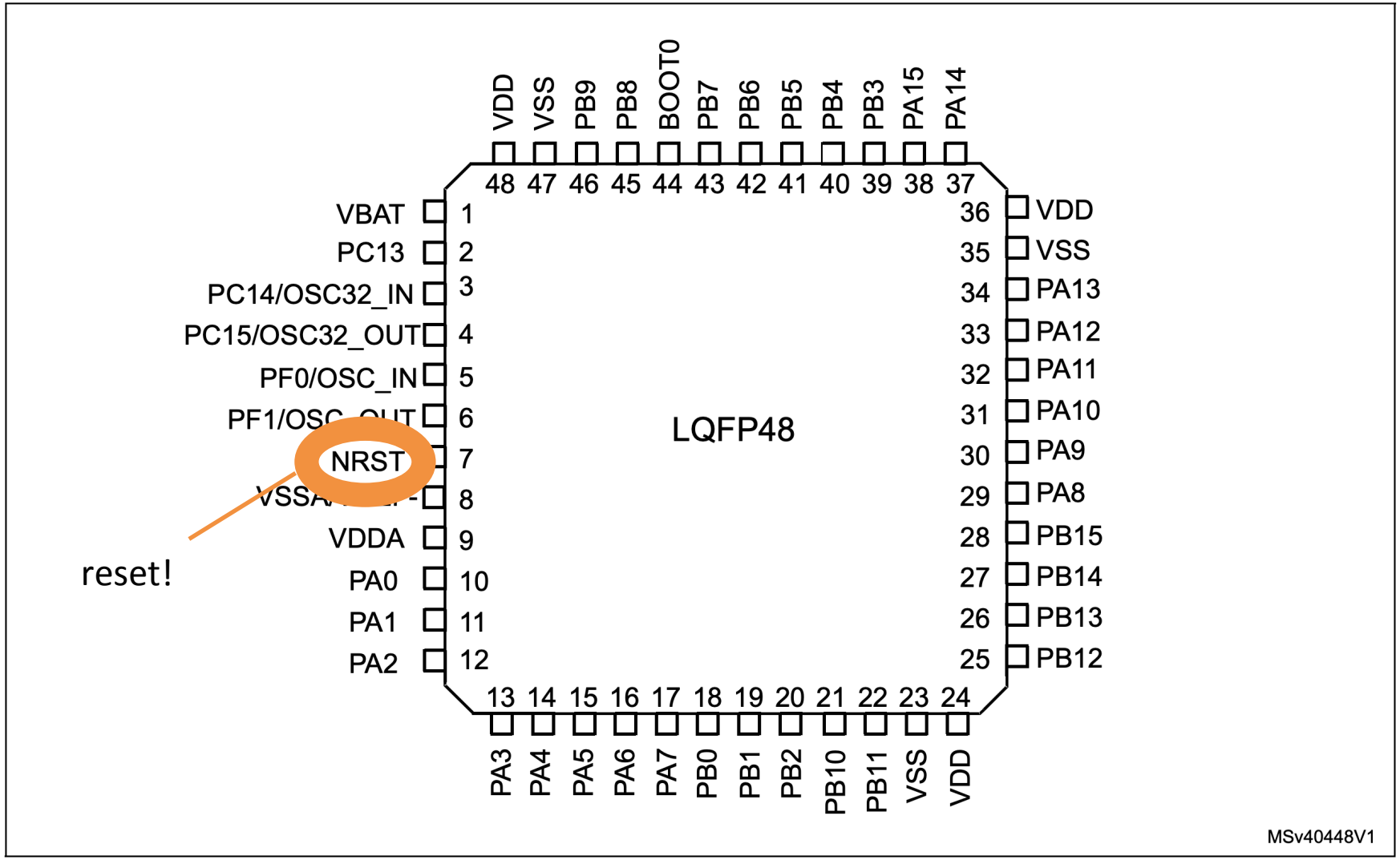
What needs to be reset?



Purging state in a CPU, attempt #1



Purging state in a CPU, attempt #2



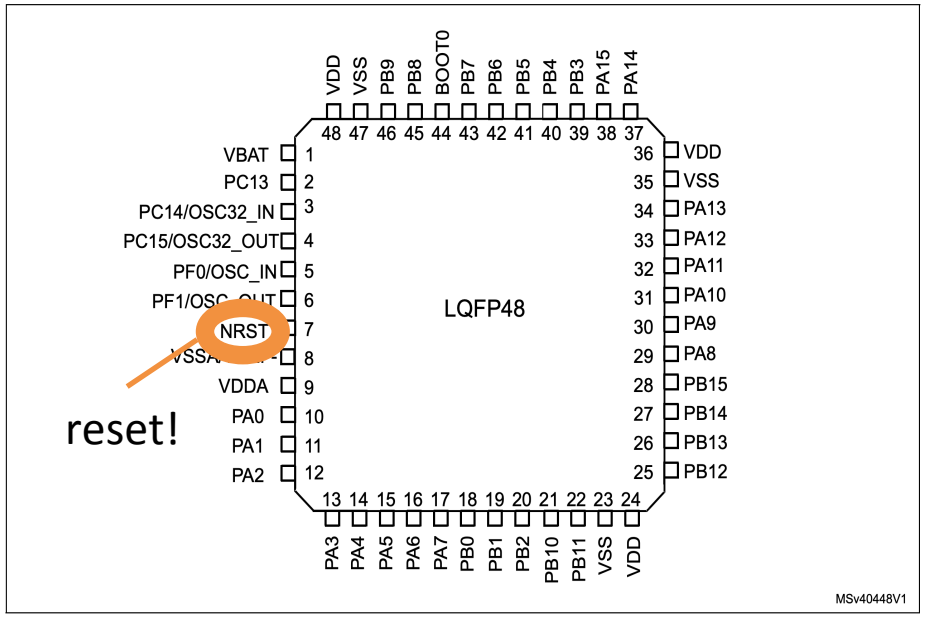
What does reset mean?

3.3 Reset

Upon reset, a hart's privilege mode is set to M. The `mstatus` fields MIE and MPRV are reset to 0, and the VM field is reset to Mbare. The `pc` is set to an implementation-defined reset vector. The `mcause` register is set to a value indicating the cause of the reset. All other hart state is undefined.

RISC-V Instruction Set Manual

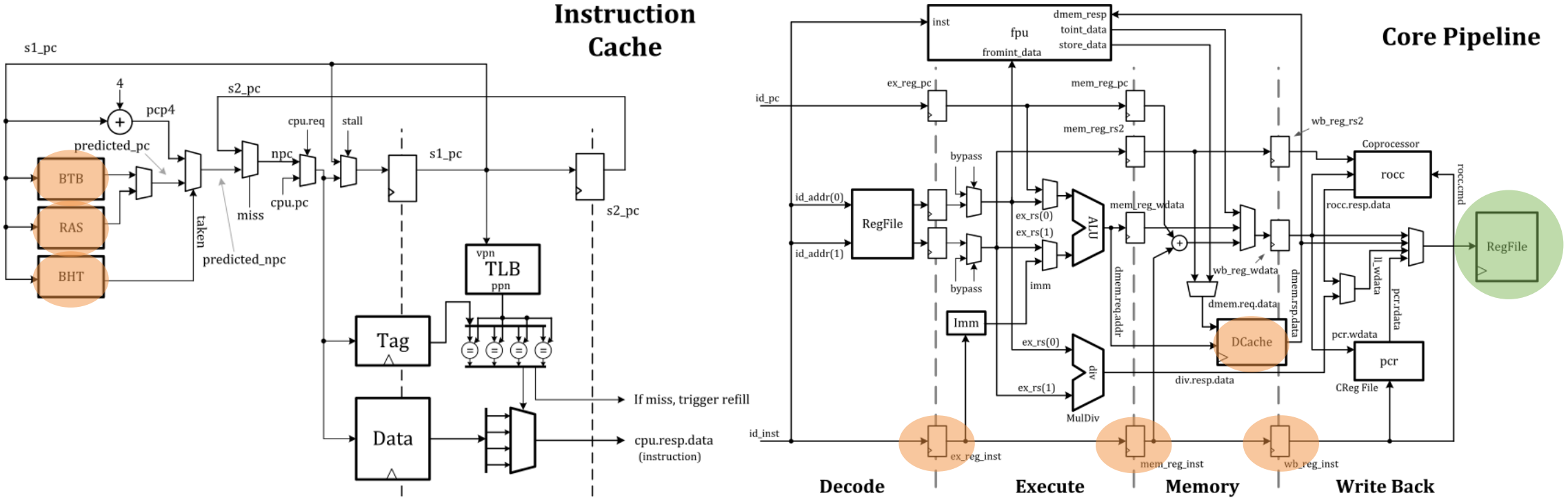
Purging state in a CPU, attempt #3



+

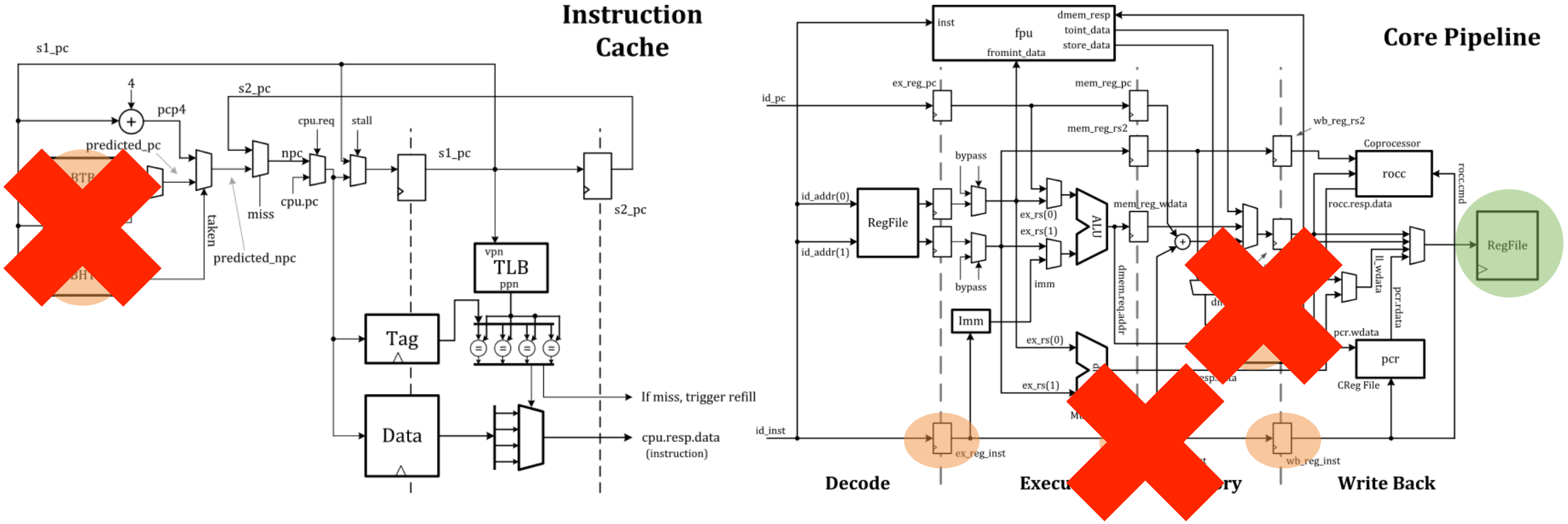
```
mov r0, #0
mov r1, #0
mov r2, #0
...
```

Architectural and micro-architectural state



Examples of architectural state vs micro-architectural state

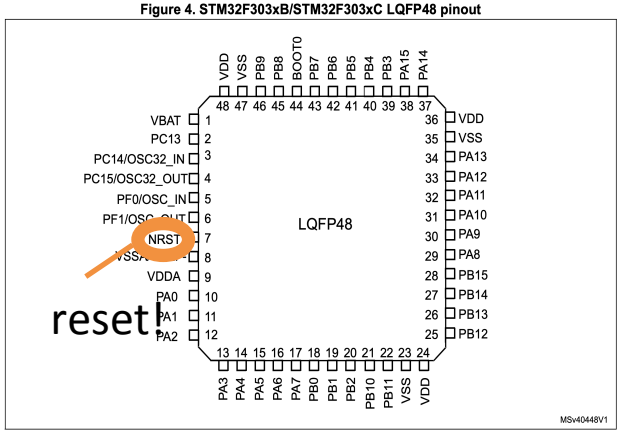
Minimizing complexity



Simpler processors have less micro-architectural state

Purging state in a CPU, final attempt

Figure 4. STM32F303xB/STM32F303xC LQFP48 pinout

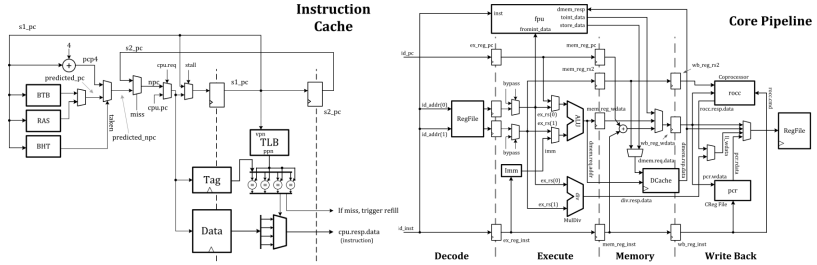


reset!

These instructions affect micro-architectural state

```
mov r0, #0
mov r1, #0
mov r2, #0
+ ...
  // do things that end up
  // clearing internal state
  ...
```

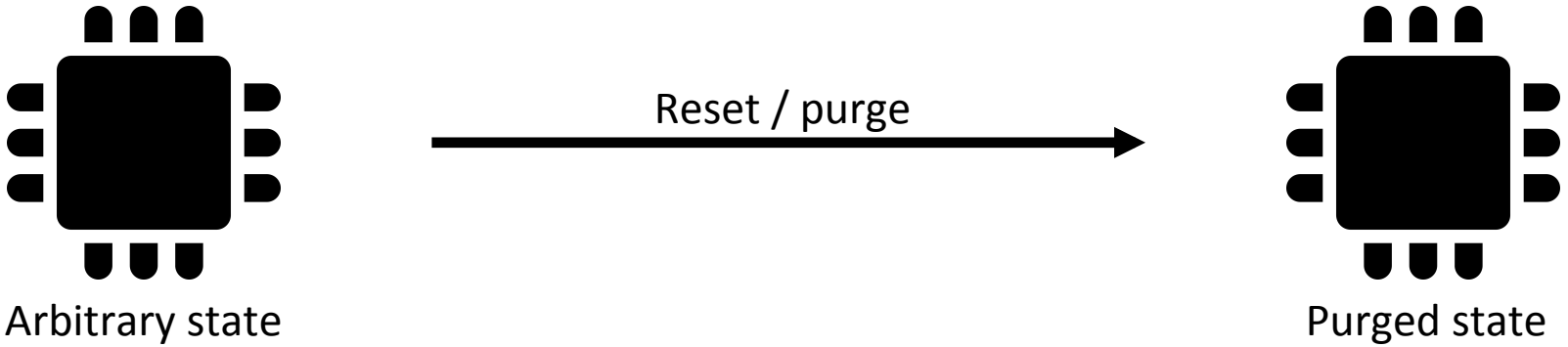
Check against CPU implementation



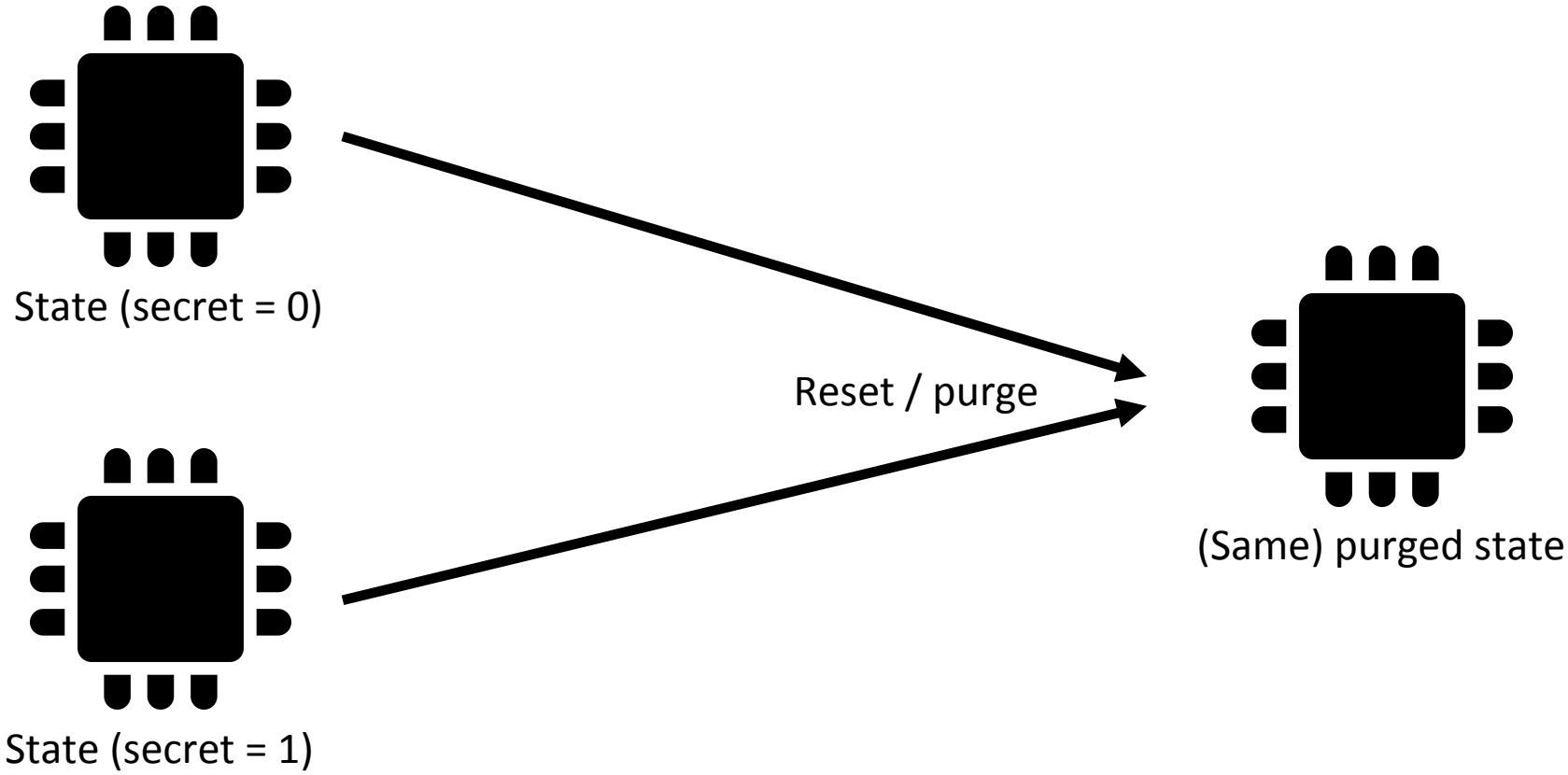

```
always @(posedge clk) begin
        if (mem_xfer) begin
                mem_rdata_q <= COMPRESSED_ISA ? mem_rdata_latched : mem_rdata;
                next_insn_opcode <= COMPRESSED_ISA ? mem_rdata_latched : mem_rdata;
        end

        if (COMPRESSED_ISA && mem_done && (mem_do_prefetch || mem_do_rinst)) begin
                case (mem_rdata_latched[1:0])
                        2'b00: begin // Quadrant 0
                                case (mem_rdata_latched[15:13])
                                        3'b000: begin // C.ADDI4SPN
                                                mem_rdata_q[14:12] <= 3'b000;
                                                mem_rdata_q[31:20] <= {2'b0, mem_rdata_latched[10:7], mem_rdata_latched
                                        end
                                        3'b010: begin // C.LW
                                                mem_rdata_q[31:20] <= {5'b0, mem_rdata_latched[5], mem_rdata_latched[1
                                                mem_rdata_q[14:12] <= 3'b 010;
                                        end
                                        3'b 110: begin // C.SW
                                                {mem_rdata_q[31:25], mem_rdata_q[11:7]} <= {5'b0, mem_rdata_latched[5],
                                                mem_rdata_q[14:12] <= 3'b 010;
                                        end
                                endcase
                        end
                        2'b01: begin // Quadrant 1
                                case (mem_rdata_latched[15:13])
                                        3'b 000: begin // C.ADDI
                                                mem_rdata_q[14:12] <= 3'b000;
                                                mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12], mem_rdata_latched
                                        end
                                        3'b 010: begin // C.LI
                                                mem_rdata_q[14:12] <= 3'b000;
                                                mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12], mem_rdata_latched
                                        end
                                        3'b 011: begin
```

How do we know that reset is correct?



How do we know that reset is correct?



Tool: Satisfiability Modulo Theories (SMT)

(x AND y) OR (NOT z) → SAT: {x = False, y = False, z = False}

(x AND y) AND ((NOT x) AND z) → UNSAT

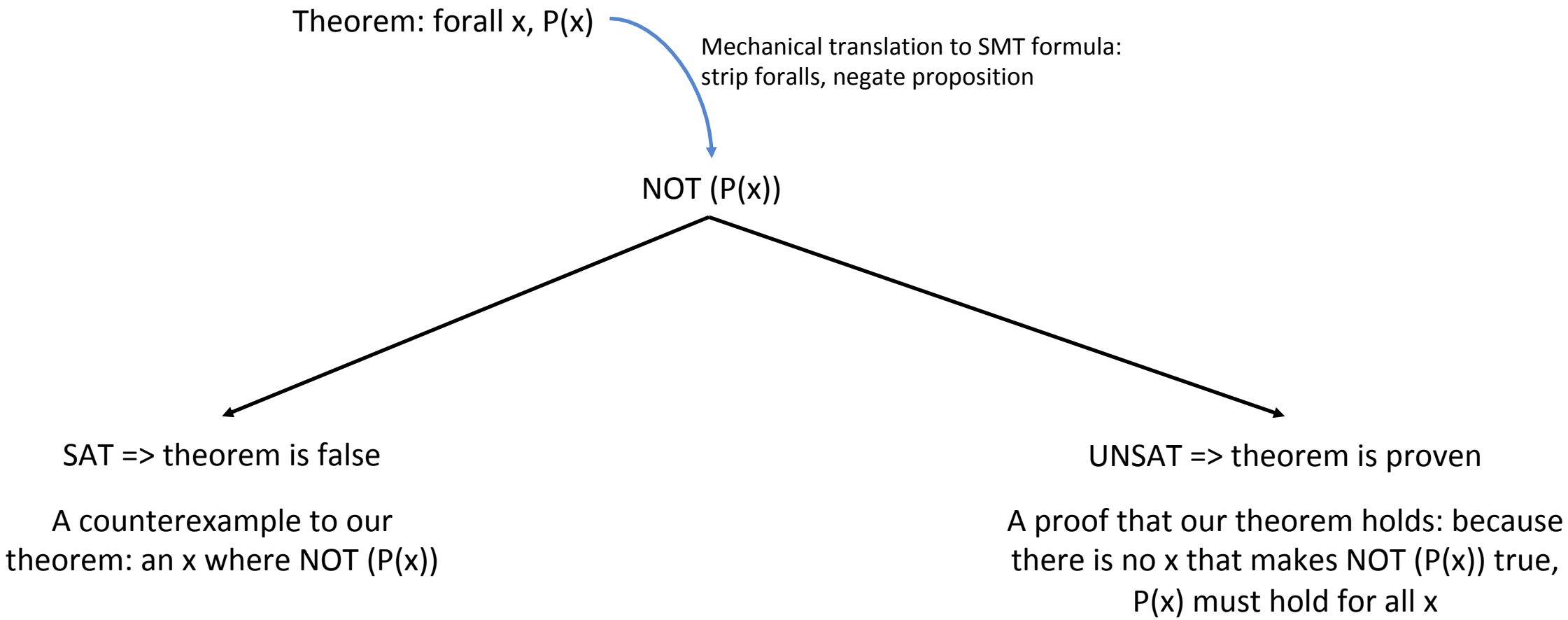
SAT solvers

SMT: SAT on steroids

x: Int, y: Int
x + y < 1 AND x + 1 = 3 AND y > 0 → UNSAT

x: BitVec(8)
x > 0 AND x + 1 < 0 → SAT: {x = 127}

SMT solvers as theorem provers



SMT solvers as theorem provers

Theorem: forall x y : Real,
min(x, y) <= (x + y)/2 <= max(x, y)

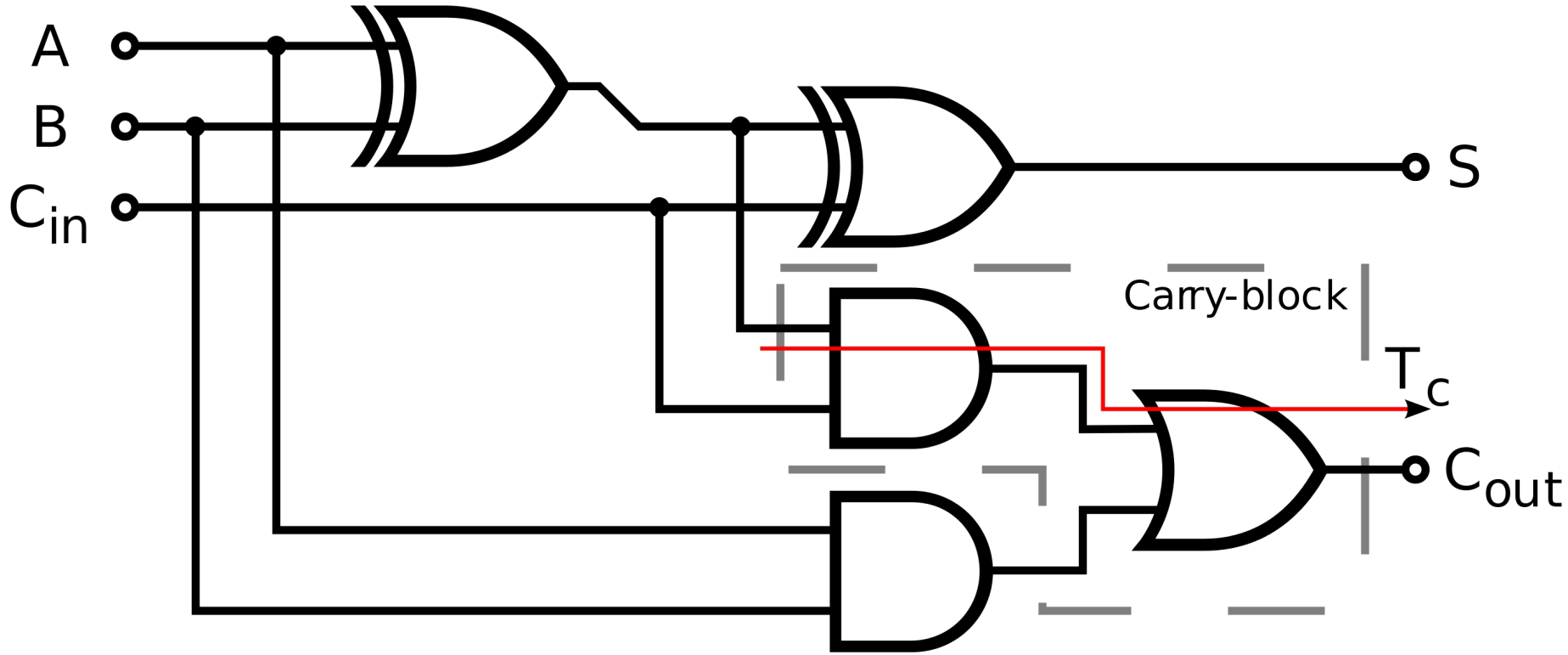
NOT [min(x, y) <= (x + y)/2 <= max(x, y)]

```
>>> from z3 import *
>>> x = Real('x'); y = Real('y'); s = Solver()
>>> Min = z3.If(x < y, x, y)
>>> Max = z3.If(x < y, y, x)
>>> avg = (x + y)/2
>>> theorem = And(Min <= avg, avg <= Max)
>>> s.add(Not(theorem))
>>> s.check()
unsat
```

Reset: theorem statement

forall s s’: CPU state,
purge(s) = purge(s’)

Combinatorial circuits



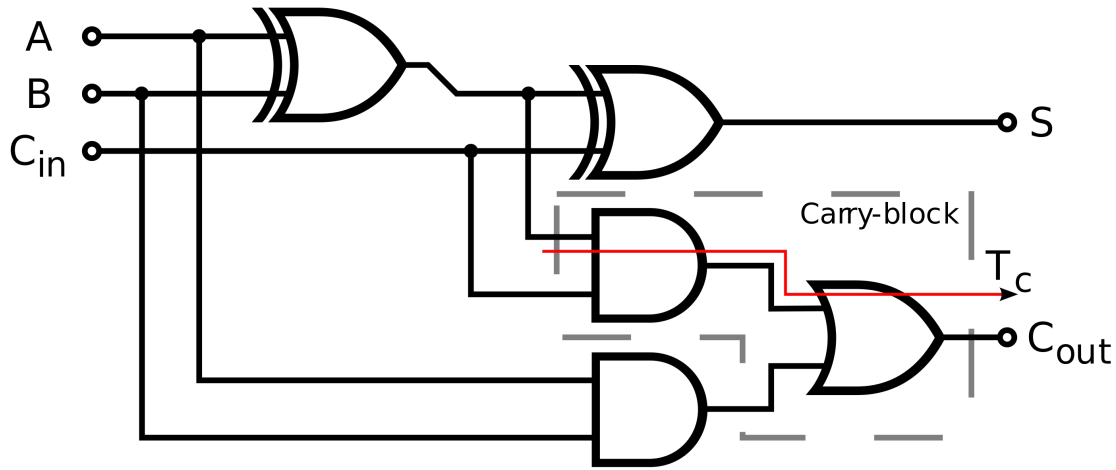
Full adder

Combinatorial circuits

```
module full_adder(input a, input b, input c_in, output s, output c_out);

    wire xor_in;
    assign xor_in = a ^ b;
    assign s = xor_in ^ c_in;
    wire and_in;
    assign and_in = a & b;
    assign c_out = (xor_in & c_in) | and_in;

endmodule
```

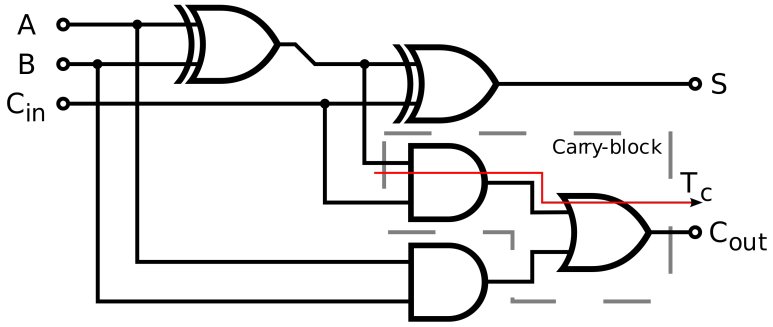


Combinatorial circuits

```
def full_adder(a, b, c_in):
    xor_in = Xor(a, b)
    and_in = And(a, b)

    s = Xor(xor_in, c_in)
    c_out = Or(And(xor_in, c_in), and_in)

    return (s, c_out)
```




```
input1 = (Bool('a1'), Bool('b1'), Bool('c_in1'))
input2 = (Bool('a2'), Bool('b2'), Bool('c_in2'))

solver = Solver()
solver.add(input1[0] == input2[1])
solver.add(input1[1] == input2[0])
solver.add(input1[2] == input2[2])
```

Preconditions:
a/b are swapped
c_in is the same

```
def all_eq(e1s, e2s):
    return And(*[e1 == e2 for e1, e2 in zip(e1s, e2s)])
```

```
solver.add(Not(all_eq(
    full_adder(*input1),
    full_adder(*input2)
)))
```

Negated theorem statement:
result of adder is different

```
solver.check() # => unsat
```

Verified!

Prove: a + b = b + a (when c_in is the same)

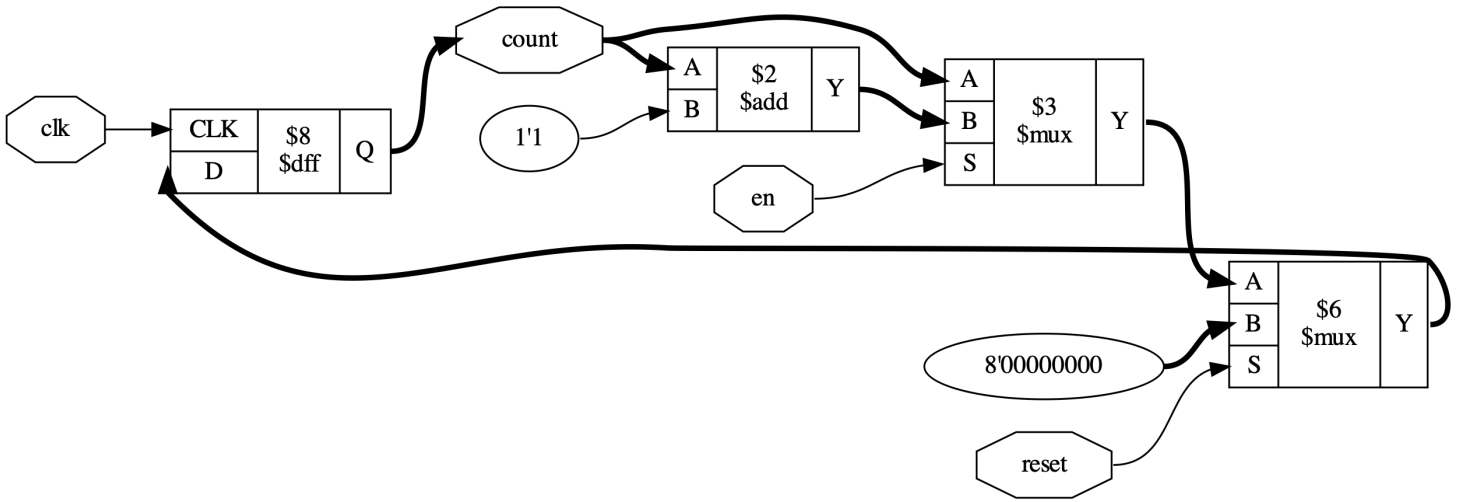
Stateful circuits

```
module counter(input clk, input reset, input en, output [7:0] count);

    reg [7:0] count;

    always @(posedge clk) begin
        if (reset) begin
            count <= 0;
        end
        else if (en) begin
            count <= count + 1;
        end
    end

endmodule
```



counter

Stateful circuits

```
counter_s = Datatype('counter_s')
counter_s.declare('counter',
        ('counter_reset', BoolSort()),
        ('counter_en', BoolSort()),
        ('counter_count', BitVecSort(8)))
reset = lambda s: counter_s.counter_reset(s)
en = lambda s: counter_s.counter_en(s)
count = lambda s: counter_s.counter_count(s)
counter_s = counter_s.create()

# transition relation
def counter_t(s1, s2):
    return If(reset(s1),
              # if reset, then count becomes 0
              count(s2) == 0,
              If(en(s1),
                  # if enabled, count increments
                  count(s2) == count(s1) + 1,
                  # otherwise, stays the same
                  count(s2) == count(s1)))
```

Proof (attempt): as long as it's not reset, count doesn't decrease

```
s1 = Function('s1', counter_s)()
s2 = Function('s2', counter_s)()

s = Solver()
s.add(counter_t(s1, s2))
s.add(Not(reset(s1)))
```

Preconditions: s1 steps to s2, counter is not reset

```
s.add(count(s2) < count(s1))
```

Negated theorem statement: count decreases

```
s.check() # => sat
m = s.model()

m.evaluate(s1)
# => counter(False, True, 127)
```

Concrete counterexample!

Converting CPU implementation to SMT

```
if (COMPRESSED_ISA && mem_done && (mem_do_prefetch || mem_do_rinst)) begin
        case (mem_rdata_latched[1:0])
                2'b00: begin // Quadrant 0
                        case (mem_rdata_latched[15:13])
                                3'b000: begin // C.ADDI4SPN
                                        mem_rdata_q[14:12] <= 3'b000;
                                        mem_rdata_q[31:20] <= {2'b0, mem_rdata_latched[10:7], me
                                end
                                3'b010: begin // C.LW
                                        mem_rdata_q[31:20] <= {5'b0, mem_rdata_latched[5], mem_r
                                        mem_rdata_q[14:12] <= 3'b 010;
                                end
                                3'b 110: begin // C.SW
                                        {mem_rdata_q[31:25], mem_rdata_q[11:7]} <= {5'b0, mem_rd
                                        mem_rdata_q[14:12] <= 3'b 010;
                                end
                        endcase
                end
                2'b01: begin // Quadrant 1
                        case (mem_rdata_latched[15:13])
                                3'b 000: begin // C.ADDI
                                        mem_rdata_q[14:12] <= 3'b000;
                                        mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12], me
                                end
                                3'b 010: begin // C.LI
                                        mem_rdata_q[14:12] <= 3'b000;
                                        mem_rdata_q[31:20] <= $signed({mem_rdata_latched[12], me
                                end
                                3'b 011: begin
                                        if (mem_rdata_latched[11:7] == 2) begin // C.ADDI16SP
                                                mem_rdata_q[14:12] <= 3'b000;
                                                mem_rdata_q[31:20] <= $signed({mem_rdata_latched
                                                        mem_rdata_latched[5], mem_rdata_
                                        end else begin // C.LUI
                                                mem_rdata_q[31:12] <= $signed({mem_rdata_latched
                                        end
                                end
                                3'b100: begin
                                        if (mem_rdata_latched[11:10] == 2'b00) begin // C.SRLI
                                                mem_rdata_q[31:25] <= 7'b0000000;
                                                mem_rdata_q[14:12] <= 3'b 101;
                                        end
```

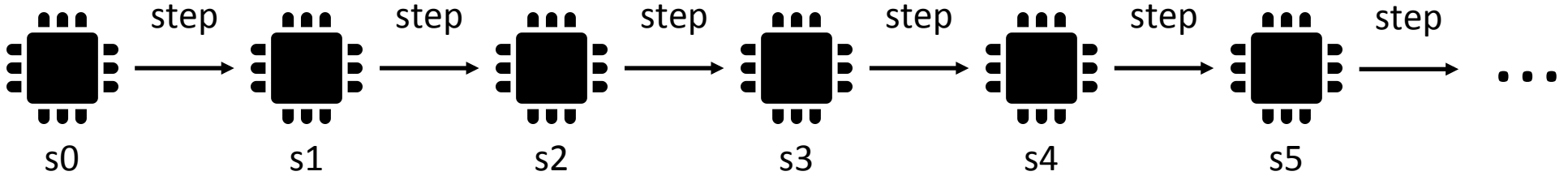
mechanical translation

```
registers.append('mem_addr')
def picorv32_n_mem_addr(state): return picorv32__84(state)
# yosys-smt2-register mem_do_prefetch 1
registers.append('mem_do_prefetch')
def picorv32_n_mem_do_prefetch(state): return (Extract(0, 0, picorv32__85(state)) == 0b1)
# yosys-smt2-register mem_do_rdata 1
registers.append('mem_do_rdata')
def picorv32_n_mem_do_rdata(state): return (Extract(0, 0, picorv32__86(state)) == 0b1)
# yosys-smt2-register mem_do_rinst 1
registers.append('mem_do_rinst')
def picorv32_n_mem_do_rinst(state): return (Extract(0, 0, picorv32__87(state)) == 0b1)
# yosys-smt2-register mem_do_wdata 1
registers.append('mem_do_wdata')
def picorv32_n_mem_do_wdata(state): return (Extract(0, 0, picorv32__88(state)) == 0b1)
# yosys-smt2-output mem_instr 1
outputs.append('mem_instr')
# yosys-smt2-register mem_instr 1
registers.append('mem_instr')
def picorv32_n_mem_instr(state): return (Extract(0, 0, picorv32__89(state)) == 0b1)
# yosys-smt2-output mem_la_addr 32
outputs.append('mem_la_addr')
def picorv32__93(state): return (picorv32__92(state) & BitVecVal(0b11111111111111111111111111111110, 32)) #
def picorv32__94(state): return And(Or((Extract(0, 0, picorv32__83(state)) == 0b1), False), Or((Extract(0, 0
def picorv32__95(state): return If(picorv32__94(state), picorv32__93(state), picorv32__91(state)) # \next_p
def picorv32__96(state): return (Extract(31, 2, picorv32__95(state)) + BitVecVal(0b00000000000000000000000000
def picorv32__97(state): return Or((Extract(0, 0, picorv32__87(state)) == 0b1), False, (Extract(0, 0, picor
def picorv32__98(state): return If(picorv32__97(state), Concat(picorv32__96(state), BitVecVal(0b00, 2)), Co
def picorv32_n_mem_la_addr(state): return picorv32__98(state)
# yosys-smt2-output mem_la_read 1
outputs.append('mem_la_read')
def picorv32__101(state): return Not(Or((Extract(0, 0, picorv32__100(state)) == 0b1),(Extract(1, 1, picorv3
def picorv32__102(state): return Or(picorv32__97(state), False, (Extract(0, 0, picorv32__86(state)) == 0b1)
def picorv32__103(state): return And(Or(picorv32__101(state), False), Or(picorv32__102(state), False)) # $l
def picorv32__104(state): return And(Or(picorv32__99(state), False), Or(picorv32__103(state), False)) # \me
def picorv32_n_mem_la_read(state): return picorv32__104(state)
# yosys-smt2-output mem_la_wdata 32
outputs.append('mem_la_wdata')
def picorv32__107(state): return (picorv32__106(state) == BitVecVal(0b10, 2)) # $procmux$3566_CMP
def picorv32__108(state): return (picorv32__106(state) == BitVecVal(0b01, 2)) # $eq$picorv32.v:1902$654_Y
def picorv32__109(state): return Not(Or((Extract(0, 0, picorv32__106(state)) == 0b1),(Extract(1, 1, picorv3
def picorv32__110(state): return If(picorv32__109(state), picorv32__105(state), If(picorv32__108(state), Co
0, picorv32__105(state)),  Concat(Extract(7, 0, picorv32__105(state)),  Concat(Extract(7, 0, picorv32__105(
def picorv32_n_mem_la_wdata(state): return picorv32__110(state)
# yosys-smt2-output mem_la_write 1
outputs.append('mem_la_write')
def picorv32__111(state): return And(Or(picorv32__99(state), False), Or(picorv32__101(state), False)) # $lo
def picorv32__112(state): return And(Or(picorv32__111(state), False), Or((Extract(0, 0, picorv32__88(state)
def picorv32_n_mem_la_write(state): return picorv32__112(state)
# yosys-smt2-output mem_la_wstrb 4
outputs.append('mem_la_wstrb')
def picorv32__113(state): return (BitVecVal(0b0001, 4) << Concat(BitVecVal(0b00, 2), Extract(1, 0, picorv32
def picorv32__114(state): return If((Extract(1, 1, picorv32__90(state)) == 0b1), BitVecVal(0b1100, 4), BitV
def picorv32__115(state): return If(picorv32__109(state), BitVecVal(0b1111, 4), If(picorv32__108(state), pi
def picorv32_n_mem_la_wstrb(state): return picorv32__115(state)
# yosys-smt2-input mem_rdata 32
inputs.append('mem_rdata')
def picorv32_n_mem_rdata(state): return picorv32__116(state)
# yosys-smt2-register mem_rdata_q 32
```

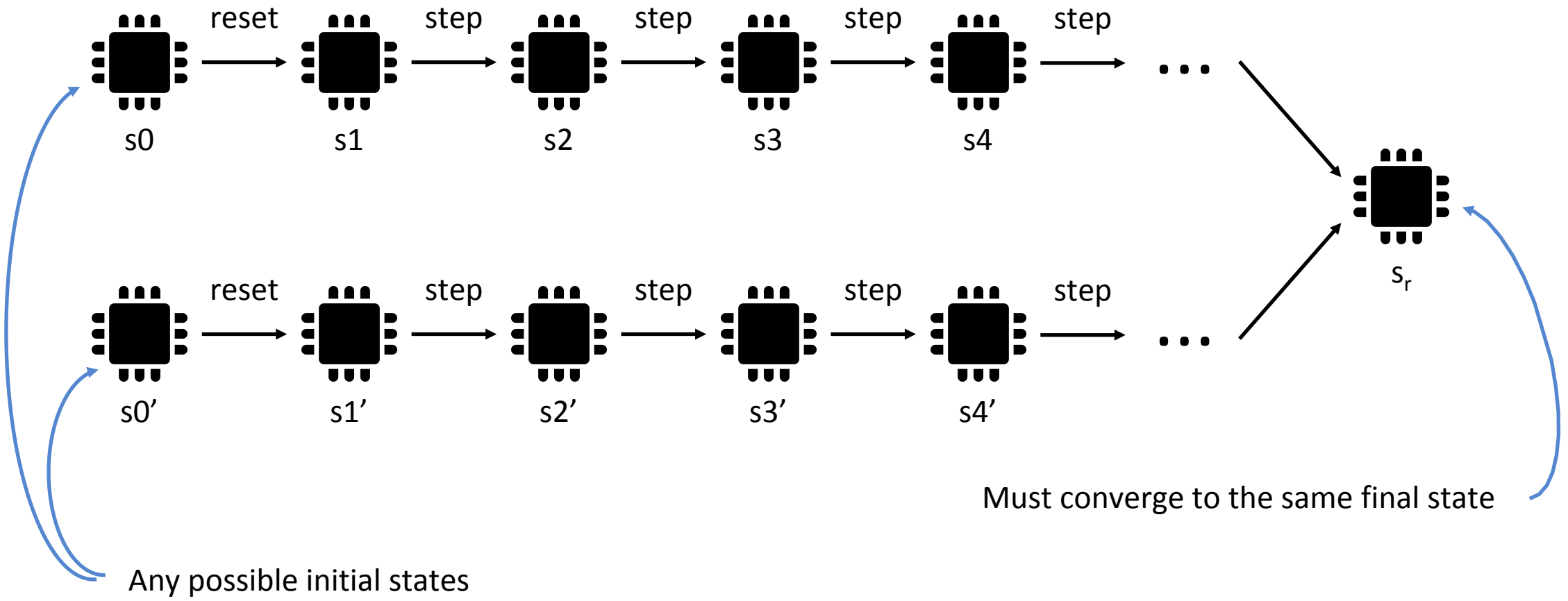
Verilog implementation:
Gate-level design of CPU

Python / Z3 SMT model:
Describes 1 cycle of CPU execution

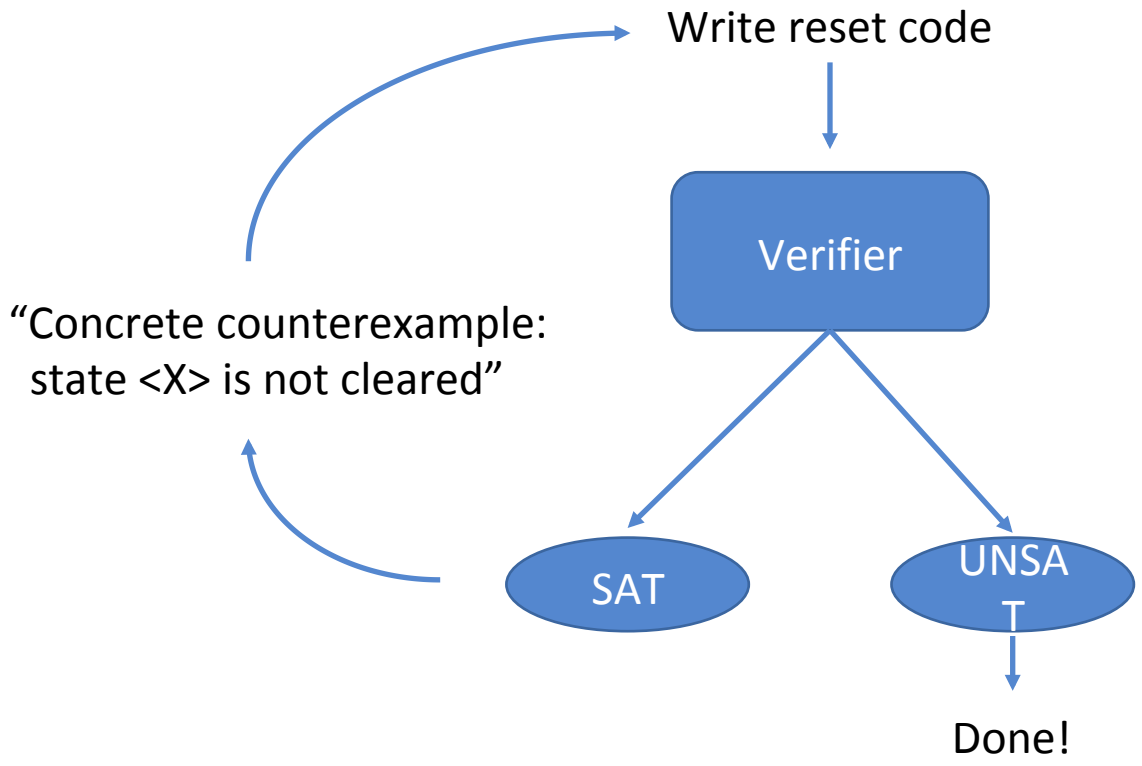
Symbolic simulation of the CPU



Proving reset correct



Interactive development of reset code



Demo (reset verification)

```
1
~
~
~
```

```
1
~
~
~
```

```
  2 nop
  1 nop
3   nop
~
~
~
```

```
 33 nop
 32 nop
 31 nop
 30 li ra,0
 29 li sp,0
 28 li gp,0
 27 li tp,0
 26 li t0,0
 25 li t1,0
 24 li t2,0
 23 li s0,0
 22 li s1,0
 21 li a0,0
 20 li a1,0
 19 li a2,0
 18 li a3,0
 17 li a4,0
 16 li a5,0
 15 li a6,0
 14 li a7,0
 13 li s2,0
 12 li s3,0
 11 li s4,0
 10 li s5,0
  9 li s6,0
  8 li s7,0
  7 li s8,0
  6 li s9,0
  5 li s10,0
  4 li s11,0
  3 li t3,0
  2 li t4,0
  1 li t5,0
34  li t6,0
~
~
~
<utf-8 | asm  100%   34:1
<, 262C written
[defcon:1.2] 1:zsh*
```

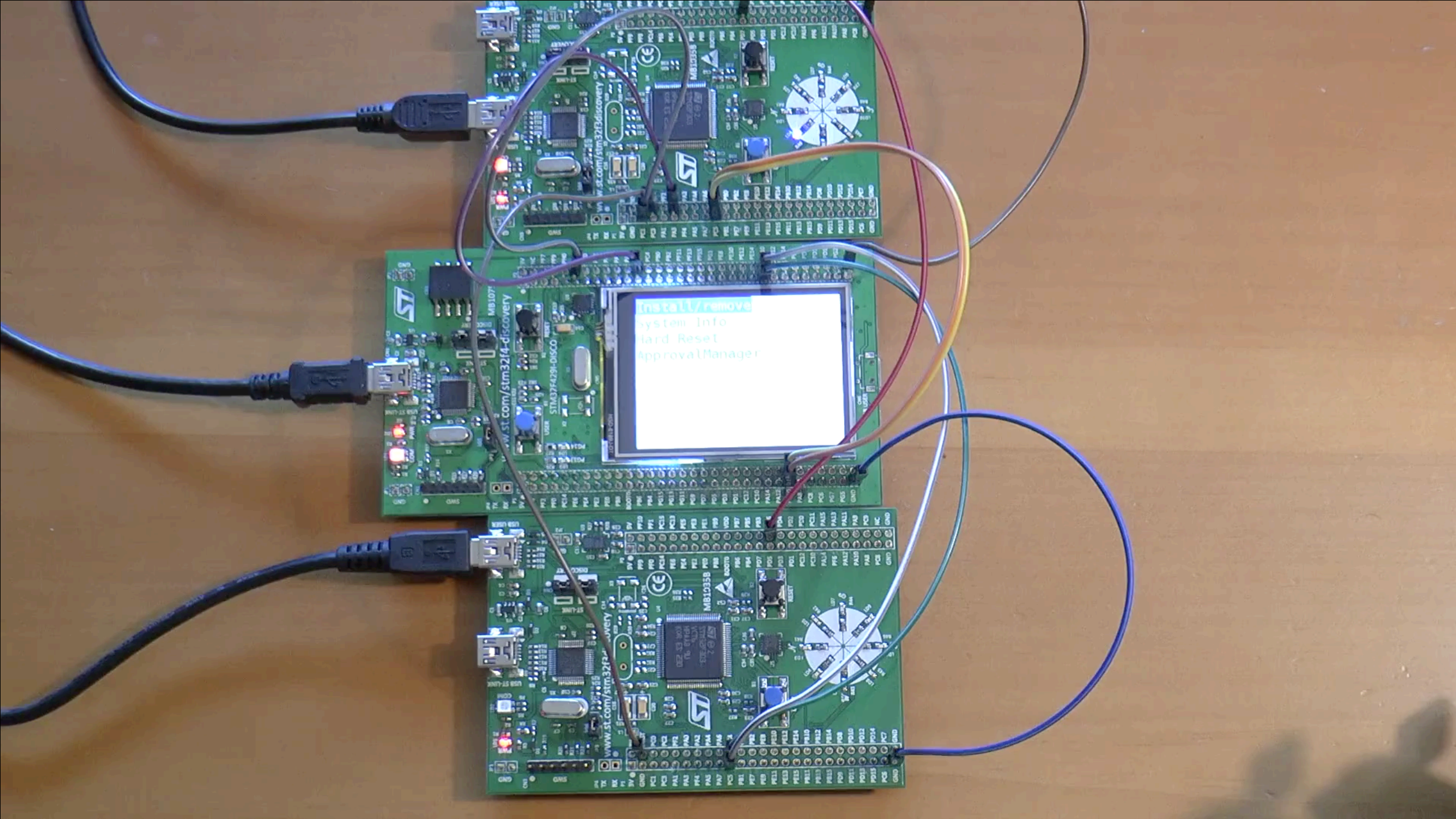
```
> 
```

```
nop
nop
nop
li ra,0
li sp,0
li gp,0
li tp,0
li t0,0
li t1,0
li t2,0
li s0,0
li s1,0
li a0,0
li a1,0
li a2,0
li a3,0
li a4,0
li a5,0
li a6,0
li a7,0
li s2,0
li s3,0
li s4,0
li s5,0
li s6,0
li s7,0
li s8,0
li s9,0
li s10,0
li s11,0
li t3,0
li t4,0
li t5,0
li t6,0
~
~
~
<utf-8 | asm  100%   34:1
<, 262C written
```

```
cycle 102
  substate cpuregs_30 is not provably reset
    (step took 2.7s)
cycle 103
  substate cpuregs_30 is not provably reset
    (step took 2.7s)
cycle 104
  substate mem_wdata is not provably reset
    (step took 1.7s)
cycle 105
  substate mem_wdata is not provably reset
    (step took 1.8s)
cycle 106
  substate mem_wdata is not provably reset
    (step took 1.7s)
cycle 107
  substate mem_wdata is not provably reset
    (step took 1.7s)
cycle 108
^C  substate mem_wdata is not provably reset
    (step took 4.4s)
cycle 109
  substate mem_wdata is not provably reset
    (step took 9.4s)
cycle 110
^CException ignored in: <function AstRef.__del__ at 0x104a26b70>
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/site-packages/z3/z3.py", line 305, in __del__
    Z3_dec_ref(self.ctx.ref(), self.as_ast())
  File "/usr/local/lib/python3.7/site-packages/z3/z3.py", line 504, in as_ast
    return Z3_sort_to_ast(self.ctx_ref(), self.ast)
  File "/usr/local/lib/python3.7/site-packages/z3/z3core.py", line 2475, in Z3_sort_to_ast
    r = _elems.f(a0, a1)
ctypes.ArgumentError: argument 2: <class 'KeyboardInterrupt'>:
^C  substate mem_wdata is not provably reset
    (step took 2.2s)
cycle 111
^C%
>
```

```
[defcon:1.2] 1:zsh*
```

Demo (hardware)


Install/remove
System Info
Hard Reset
ApprovalManager

NamedManager

logged on as anish | options | logout

Overview | Changelog | Domains/Zones | Name Servers | Configuration | User Management

OVERVIEW

Welcome to NamedManager, a PHP web-based DNS management interface for managing DNS zones.

NamedManager is licensed under the AGPL

Version 1.9.0

Conclusion

”Turning it off and on again” (with a little more details) can be used as a primitive for achieving isolation.

- **Users**: Start using transaction authorization devices
- **Developers**:
  - Support factoring out approval decisions (see WebAuthn)
  - Use verification as a tool to improve security

Code: git.io/shiva