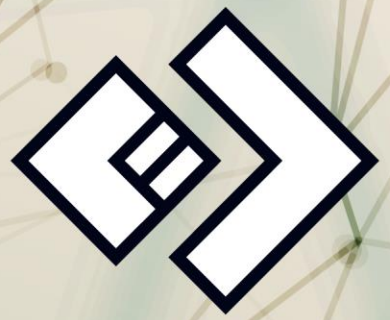


LUBISEDA

Property Generator:

Simple Generation of Formal Assertion IP

Tobias Ludwig, CEO

Gen AI and Coding Tasks

Software domain

From just an auto-complete tool to bringing up a web app in minutes

Not true for RTL code generation … yet

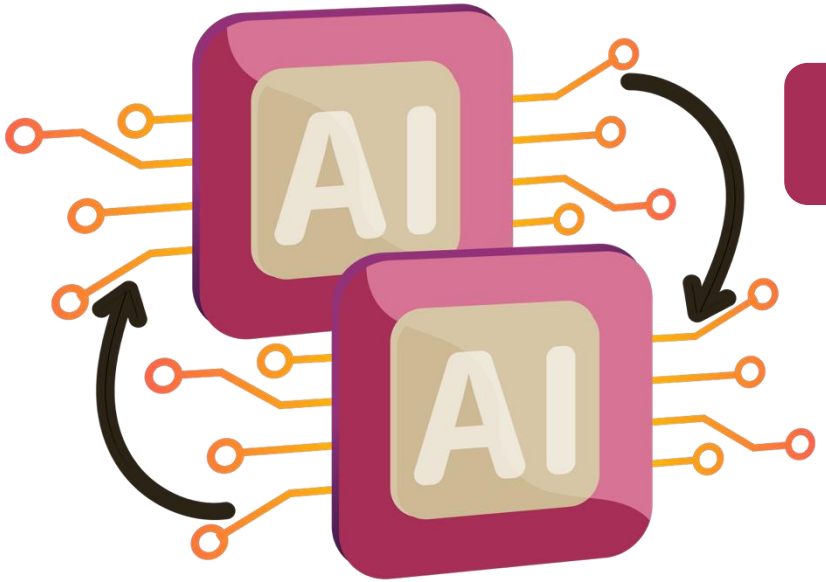
- Massive progress on VerilogEval benchmark, from 26% GPT3.5 to 63% GPT4o to 98% Agentic systems
- CVDP currently hoovers around 30%

Gen AI and RTL Design

Verification remains the bottleneck

Why not let AI verify AI?

Pursued by many … the dream of AI maximalists



Will not scale for chip design … why?

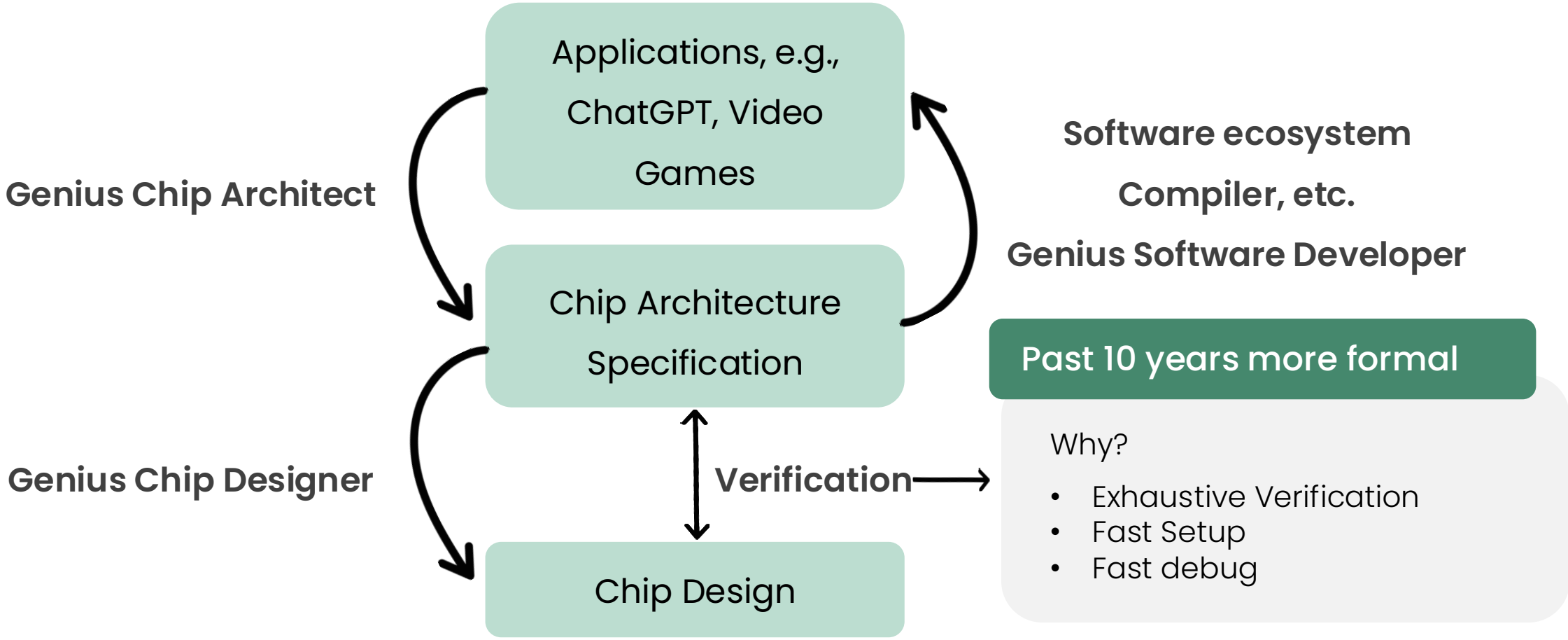
- Frontier LLMs are trained with similar data and reward functions
- They tend to make similar errors
- Amplifies the risk of masking bugs

- Bugs are expensive, re-spins costs millions of dollars
- **Human oversight will remain** most for foreseeable future
- Auditing thousands of lines of code can be more effort than writing them

Design can be creative…but verification must be deterministic, something AI is not good at!

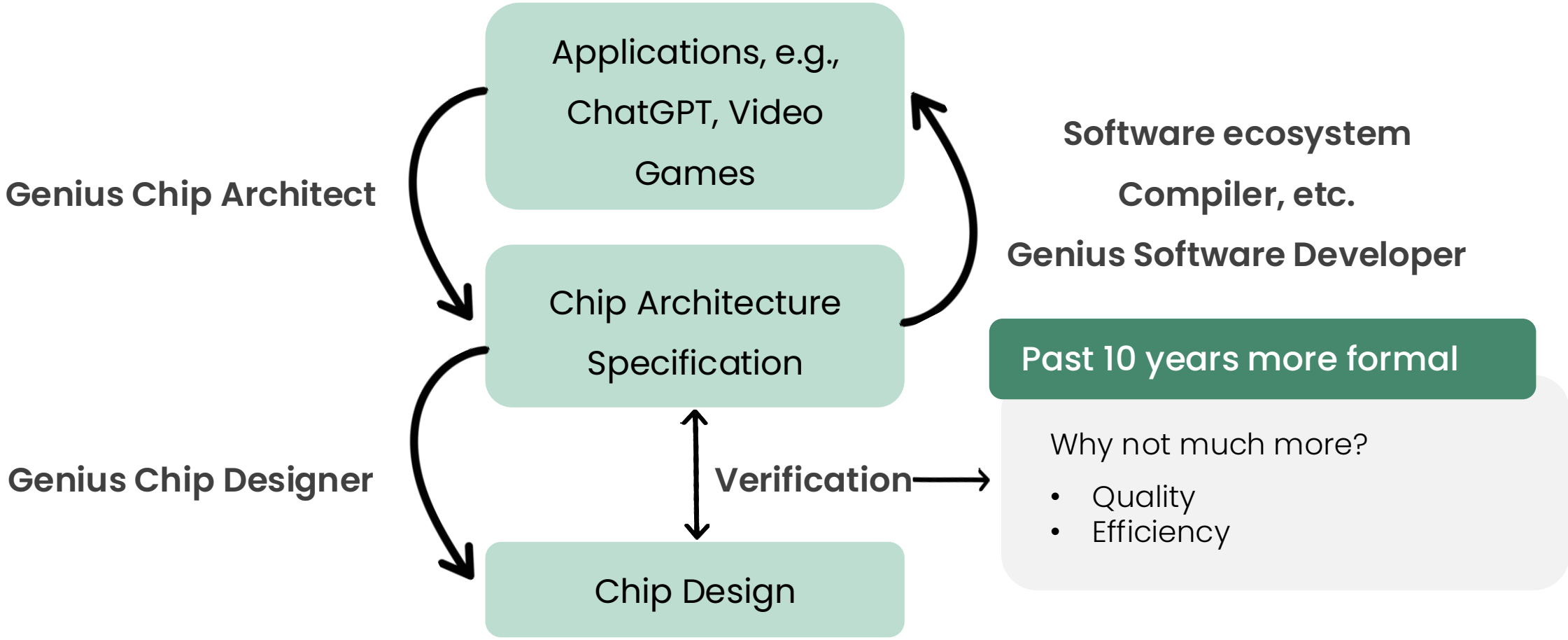
What it takes to have a successful chip?

Tens of thousands of engineer days



What it takes to have a successful chip?

Tens of thousands of engineer days



LUBIS EDA the largest independent Formal-Only player

Industrialized Formal Sign-Off

325+
Projects Delivered
Proven track record

37+
Full-time
Deep specialization, all in Germany

900+
Design Bugs Uncovered
Post UVM

4
Fortune25 clients
Trusted by world's largest players

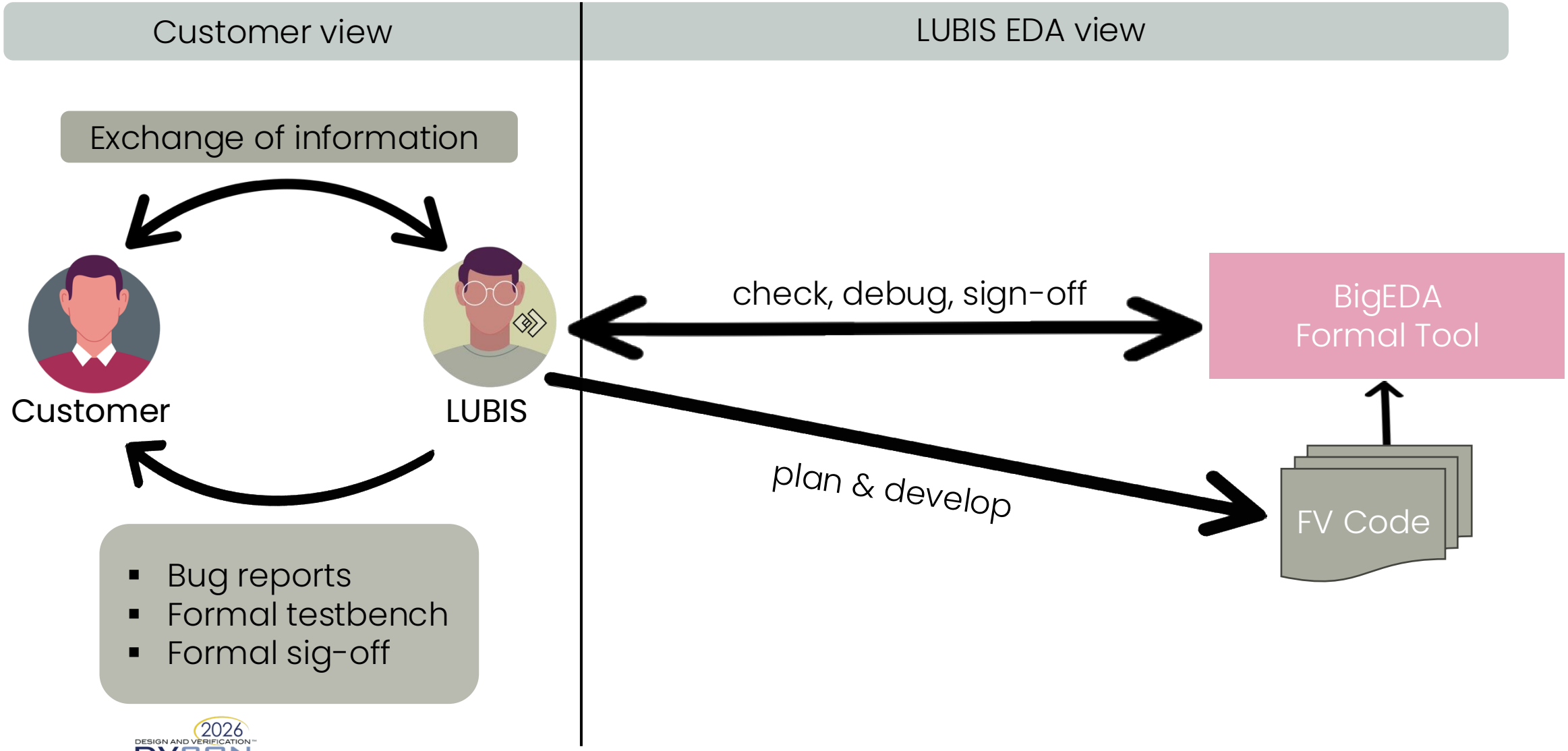
15+
Repeat Customers
Long-term partnerships

3
BIG EDA Connections
Cadence, Synopsys, Siemens

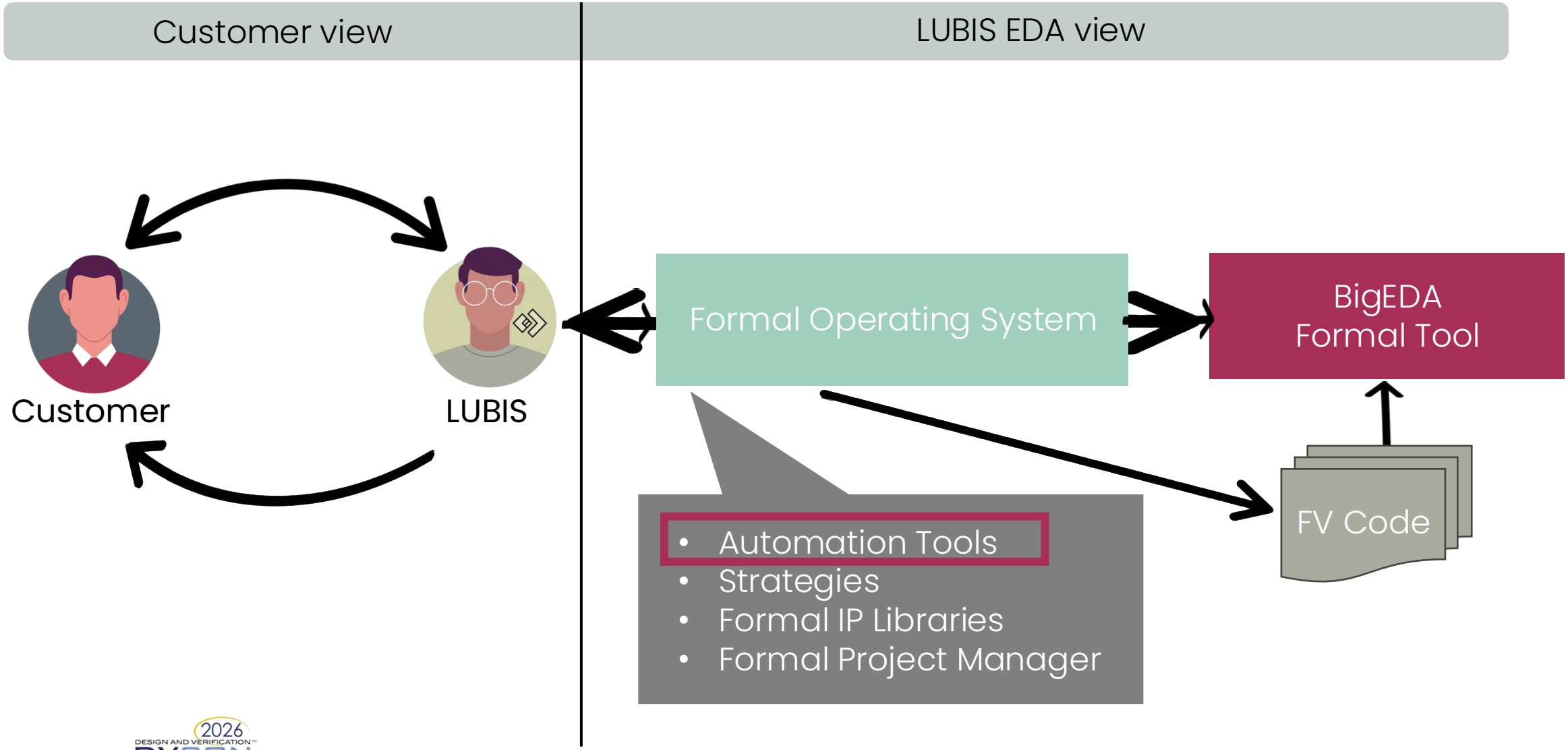
87%
Delivery on-time
Consistent performance over rolling 12 months

6+
Years
Company Record

From good old consulting

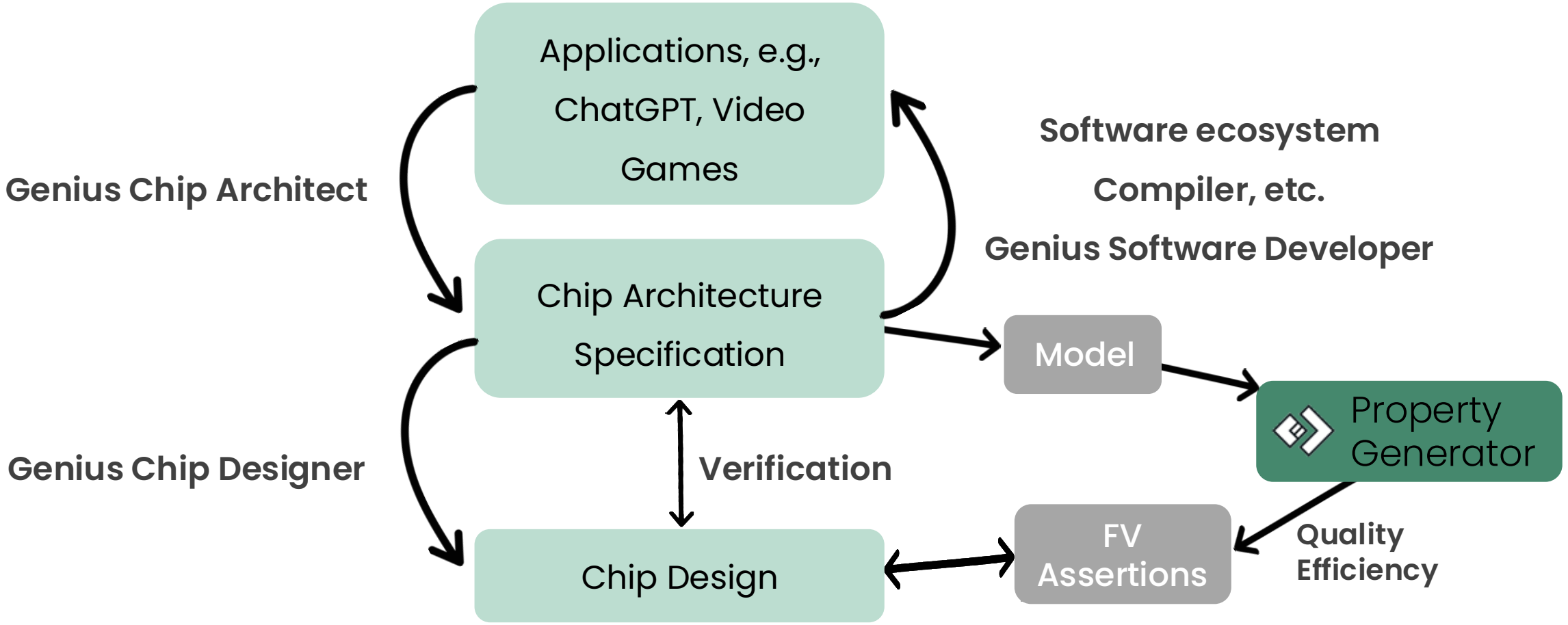


From good old consulting



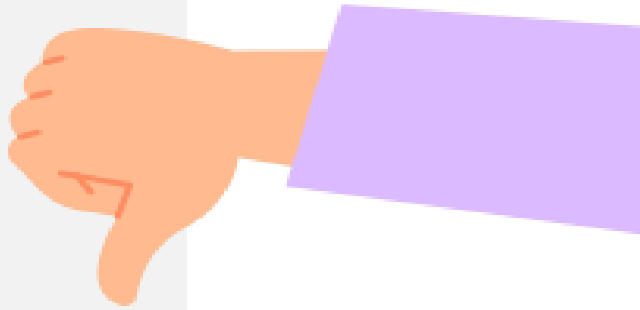
What it takes to have a successful chip?

Tens of thousands of engineer days



Mind the gap

```
1. // FIFO watermark comparator context
2. // fill_level: current FIFO occupancy (0..DEPTH)
3. // watermark: programmable threshold
4. // above_wm: 1 when fill_level > watermark
5. // below_wm: 1 when fill_level < watermark
6.
7. above_a: assert property (disable iff(!rst_n)
8.       (fill_level > watermark) |-> (above_wm && !below_wm)
9. );
10.
11. below_a: assert property (disable iff(!rst_n)
12.          (fill_level < watermark |-> (!above_wm && below_wm)
13. );
```



Complexity sneaks in fast

```
1. // NoC ingress flow-control with four throttle modes
2. // q_depth: 0..MAX
3. // t0 < t1 < t2: programmable thresholds
4. // mode: 2'b00=PASS, 2'b01=SLOW1, 2'b10=SLOW2, 2'b11=DROP
5.
6.   pass_a: assert property (disable iff(!rst_n)
7.       q_depth < t0 |-> (mode == 2'b00)
8. );
9.
10. slow1_a: assert property (disable iff(!rst_n)
11.      ((q_depth > t0) && (q_depth < t1)) |-> (mode == 2'b01)
12. );
13.
14. slow2_a: assert property (disable iff(!rst_n)
15.      ((q_depth > t1) && (q_depth < t2)) |-> (mode == 2'b10)
16. );
17.
18. drop_a: assert property (disable iff(!rst_n)
19.      (q_depth > t2) |-> (mode == 2'b11)
20. );
```

Turning intend into properties

```
1. int fill_level, watermark;
2. bool above_wm, below_wm;
3. if (fill_level > watermark) {
4.       above_wm = true;
5.       below_wm = false;
6. } else {
7.       above_wm = false;
8.       below_wm = true;
9. }
```

```
1. // FIFO watermark comparator context
2. // fill_level: current FIFO occupancy (0..DEPTH)
3. // watermark: programmable threshold
4. // above_wm: 1 when fill_level > watermark
5. // below_wm: 1 when fill_level <= watermark
6.
7. above_a: assert property (disable iff(!rst_n)
8.       (fill_level > watermark) |-> (above_wm && !below_wm)
9. );
10.
11. below_a: assert property (disable iff(!rst_n)
12.          (fill_level <= watermark |-> (!above_wm && below_wm)
13. );
```

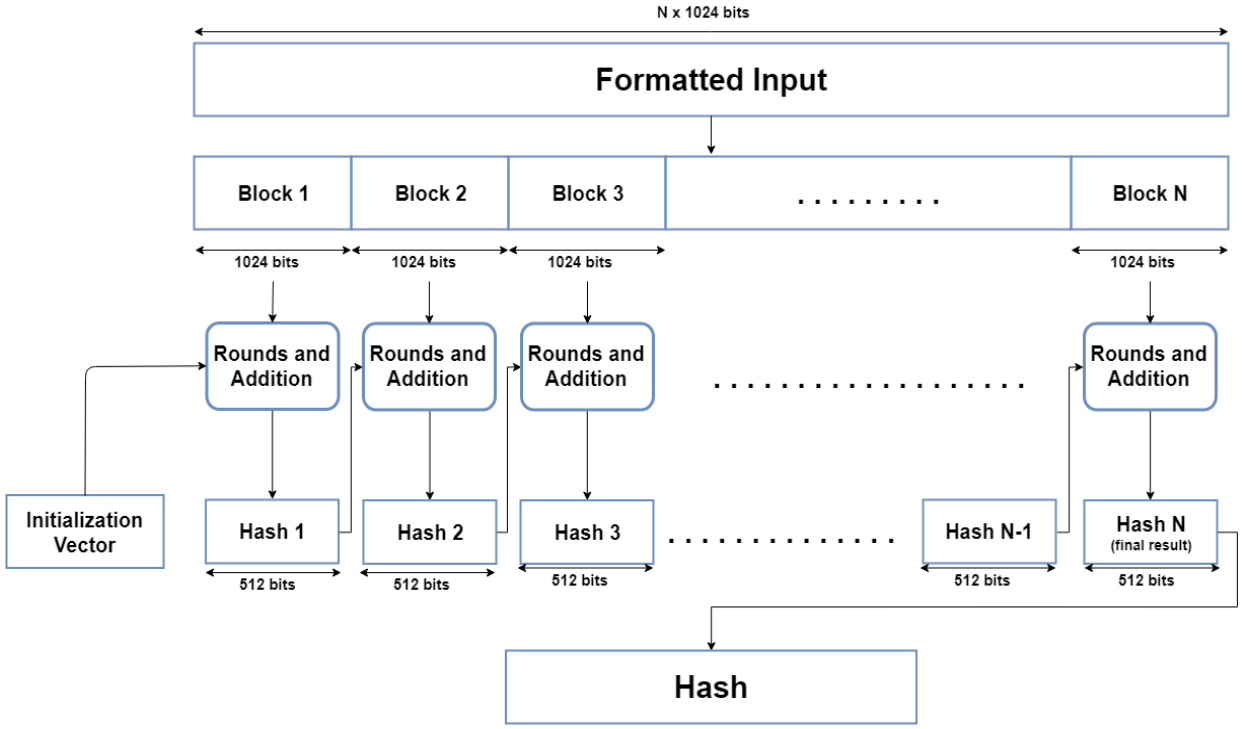
Example for today

Commercial project we did for Google

Caliptra:
SHA512 GitHub

SHA512 Algorithm

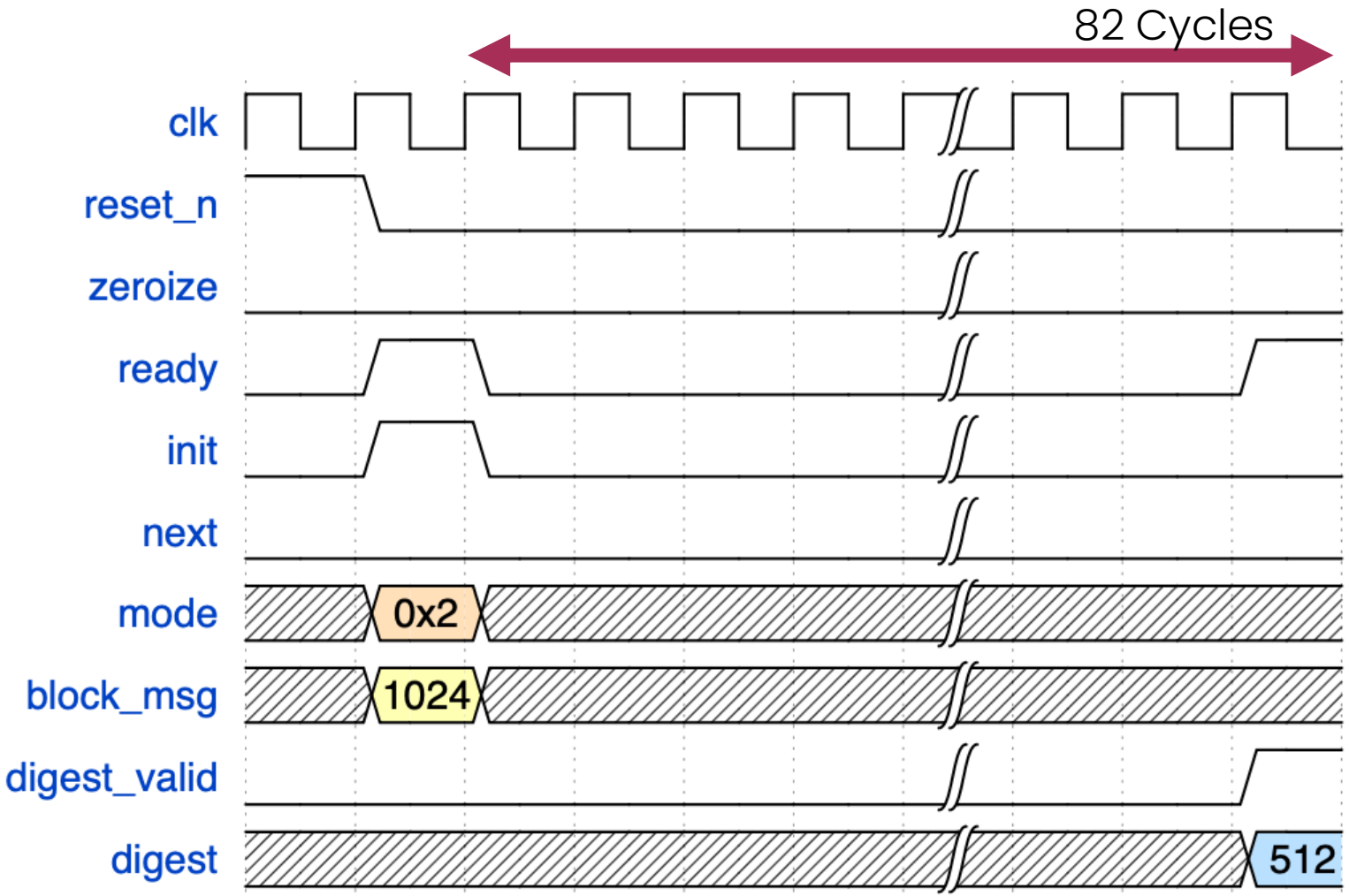
```
1. for (i=0; i<NUM_ROUNDS; ++i) {
2.         k = K[i];
3.         if (i < 16){
4.               w = W[i];
5.         } else {
6.               tmp_w = compute_w(W[14],W[9],W[1],W[0]);
7.               for (j=0; j<15; ++j) {
8.                     W[j] = W[(j+1)];
9.               }
10.              W[15] = tmp_w;
11.              w = tmp_w;
12.        }
13.
14.        t1 = T1(e, f, g, h, K[i],w);
15.        t2 = T2(a, b, c);
16.        h = g; g = f;
17.        f = e; e = (d + t1);
18.        d = c; c = b;
19.        b = a; a = (t1 + t2);
20. }
```



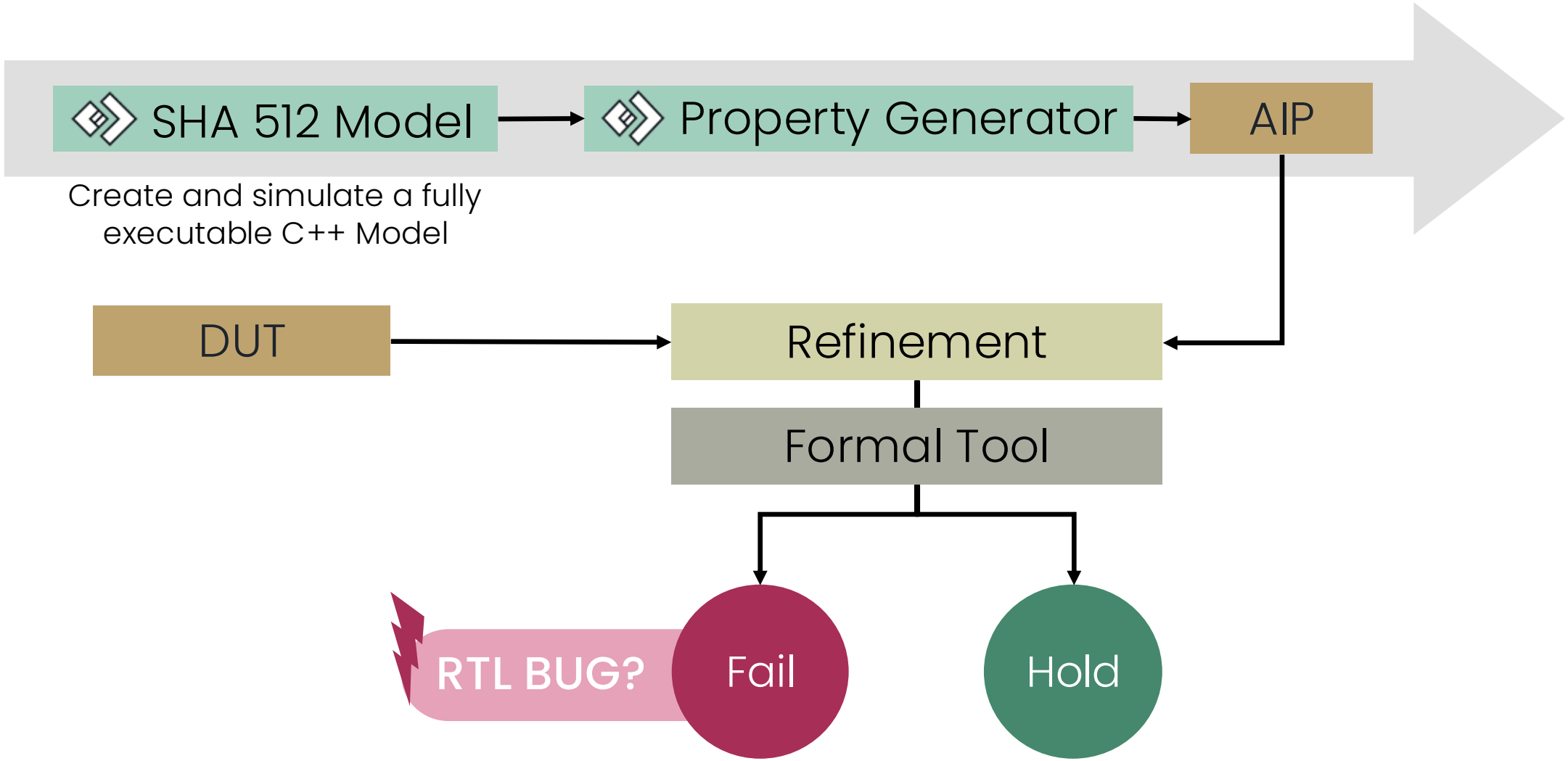
Source: https://komodoplatform.com/en/academy/sha-512/

Design Interface

From input to output

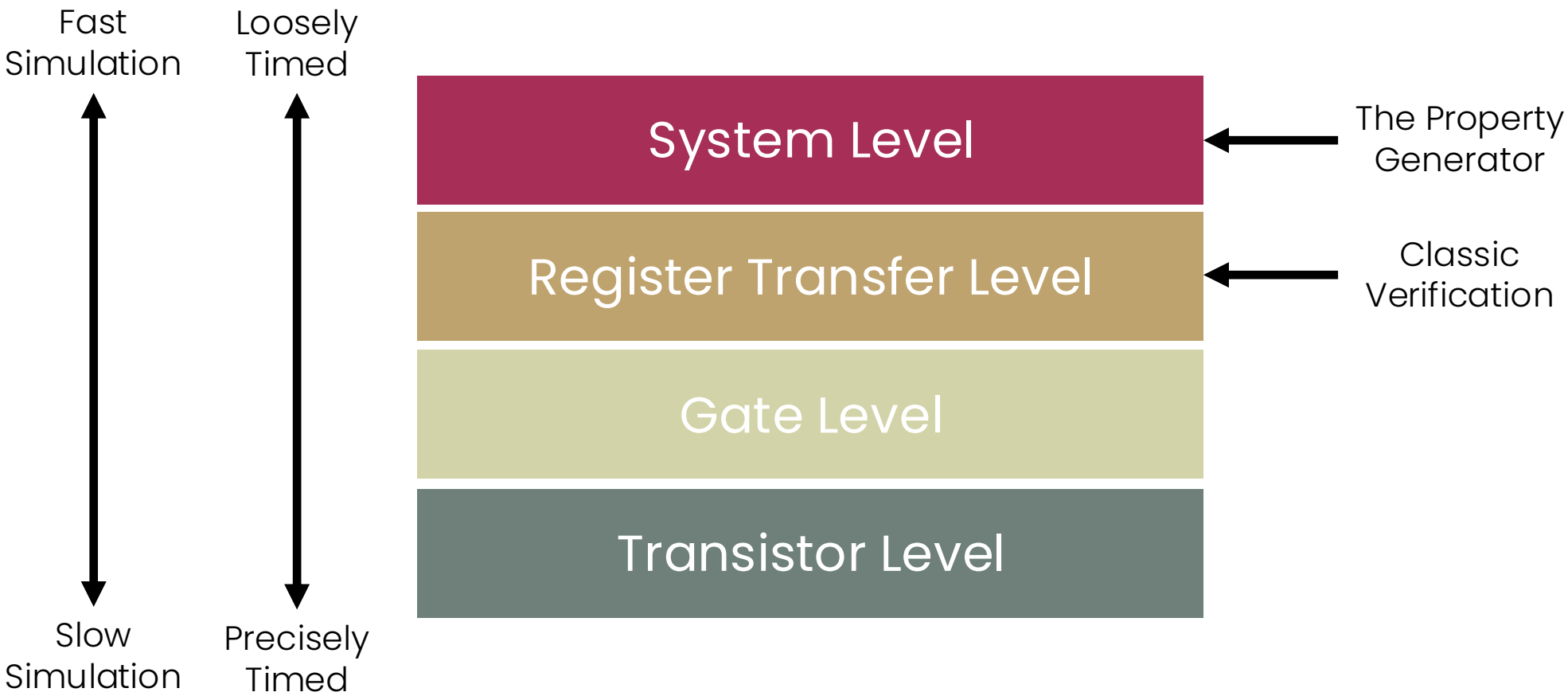


Here is what we are going to do



Abstraction Levels

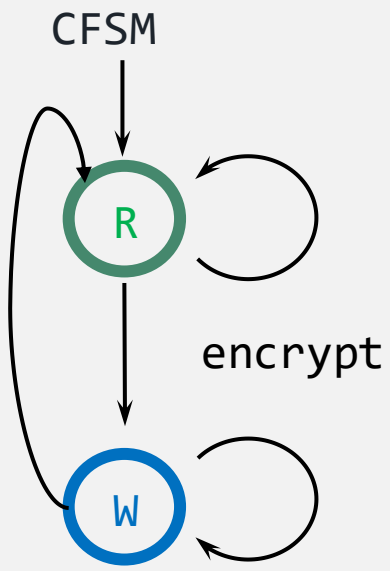
Where do we stand with the Property Generator



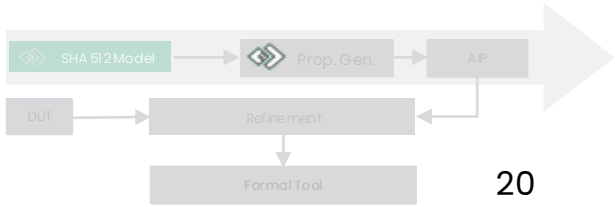
Modelling

Conceptualize the abstract behavior of the design as a model

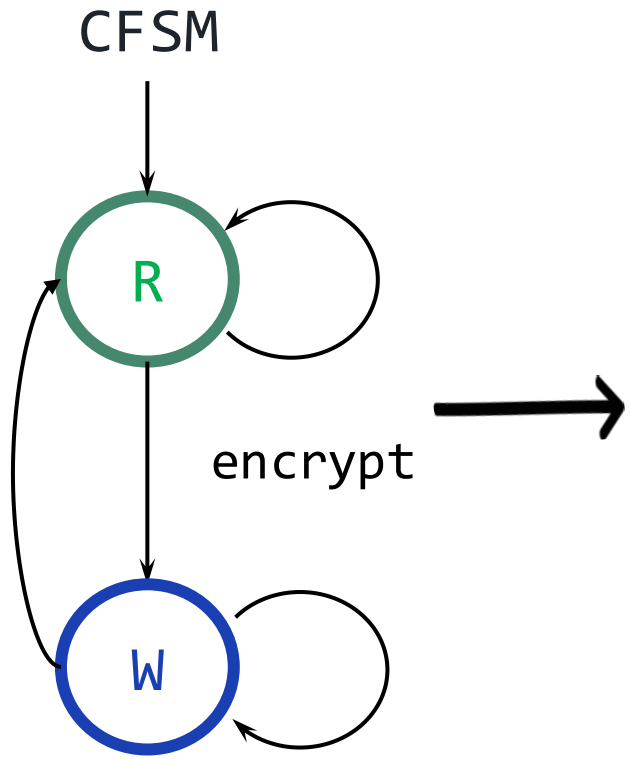
```
SC_MODULE(encrypt) {
    // constructor and declarations
    void fsm() {
       while(true){
        input→read(data);
        output→write(encrypt(data));
        // some code
    } } };
```




```
Property encrypt;
assume:
  at t:  state == R;
  at t:   data_valid == 1;
prove:
  at t+n: out(enc(data@t));
  at t+n: state == W;
endproperty;
```



Generating properties without gaps



FSM Path	Property in Assertion IP
Reset -> Read	reset_p
Read -> Read	read_to_read_p
Read -> Write	read_to_write_p
Write -> Write	write_to_write_p
Write -> Read	write_to_read_p

Model to Property

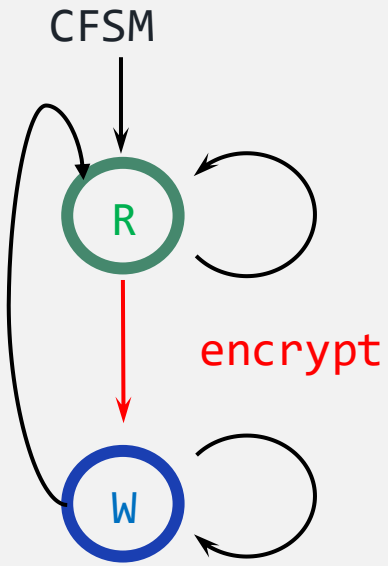
```
1. SHA_Input->read(SHA_in, “IDLE”);
2.
3. if(init){
4.   // Some Computation
5. }
6. // Some Computations
7. for(i = 0; i < NUM_ROUNDS; ++i){
8.       // Some Computation
9.       t1 = T1(e, f, g, h, K[i], w);
10.          t2 = T1(a, b, c);
11.          h = g; g = f; f = e; e = (d + t1);
12.          d = c; c = b; b = a; a = (t1 + t2);
13. }
14. insert_state(“DONE”);
```

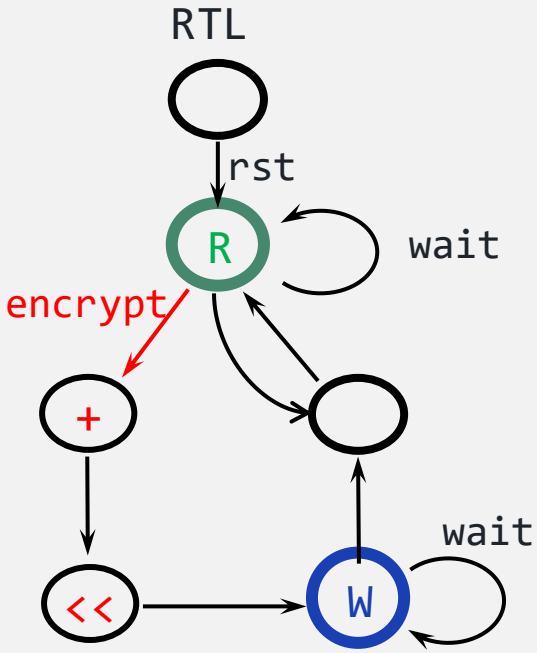
```
1. property fsm_IDLE_to_DONE_4_p;
2.    IDLE &&
3.  SHA_Input_vld &&
4.    !SHA_Input.init
5. |->
6.  ##1 ((SHA_Input_rdy == 0) &&
7.      (out_vld == 0))[*81] and
8.  ##81
9.  DONE &&
10. H == $past(H, 81) &&
11. a == ($past(T1_399_i, 81) + $past(T2_399_i, 81)) &&
12. b == ($past(T1_398_i, 81) + $past(T2_398_i, 81)) &&
13. c == ($past(T1_397_i, 81) + $past(T2_397_i, 81)) &&
14. d == ($past(T1_396_i, 81) + $past(T2_396_i, 81)) &&
15. e == (($past(T1_395_i, 81) + $past(T2_395_i, 81)) +
                                                $past(T1_399_i, 81)) &&
16. f == (($past(T1_394_i, 81) + $past(T2_394_i, 81)) +
                                                $past(T1_398_i, 81)) &&
17. g == (($past(T1_393_i, 81) + $past(T2_393_i, 81)) +
                                                $past(T1_397_i, 81)) &&
18. h == (($past(T1_392_i, 81) + $past(T2_392_i, 81)) +
                                                $past(T1_396_i, 81));
19. endproperty
```

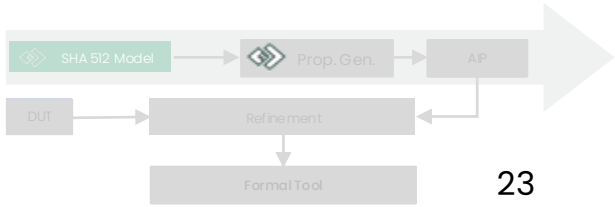
Abstraction to Implementation

Match the model with the concrete implementation

```
SC_MODULE(encrypt) {
    // constructor and declarations
    void fsm() {
       while(true){
        input→read(data);
        output→write(encrypt(data));
        // some code
    } } };
```







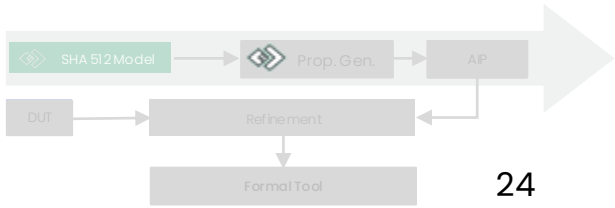
Refining the model

Provide the details missing in the abstract model, like timing and bit-accuracy

```
property encrypt(length);
      //local variables
      logic[1023:0] data_0;
      // store values @t
      ##0 (valid_data, data_0 = bus_in_data()) and
      // triggers
      ##0 state() == R
|->
      ##length state() == W and
      ##length bus_out() == enc(data_0)
endproperty;
```

```
function logic[1023:0] bus_in_data()
      return data_in;
endfunction;
```

```
function logic[1023:0] bus_in_data()
  return {
      $past(data_in,3),
      $past(data_in,2),
      $past(data_in,1),
      data_in};
endfunction;
```

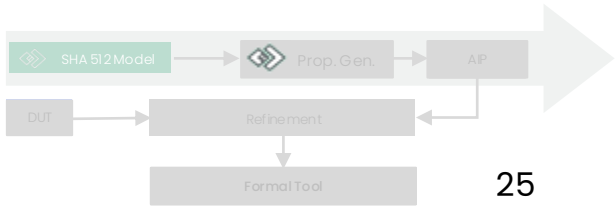


Refining the model

Provide the details missing in the abstract model, like timing and bit-accuracy

```
property encrypt(length);
      //local variables
      logic[1023:0] data_0;
      // store values @t
      ##0 (valid_data, data_0 = bus_in_data()) and
      // triggers
      ##0 state() == R
|->
      ##length state() == W and
      ##length bus_out() == enc(data_0)
endproperty;
```

```
function logic[1:0] state()
      if(state_reg == 2'b00)
          return R
      else if(state_reg == 2'b10)
          return W
endfunction;
```



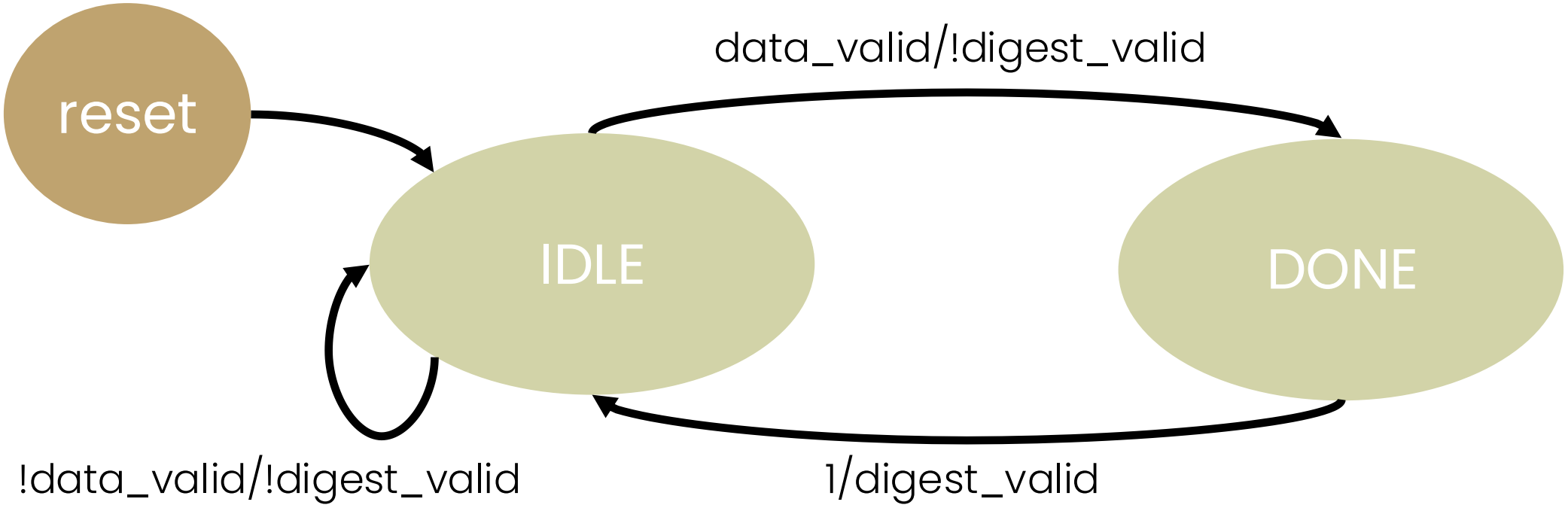
Tech Demo

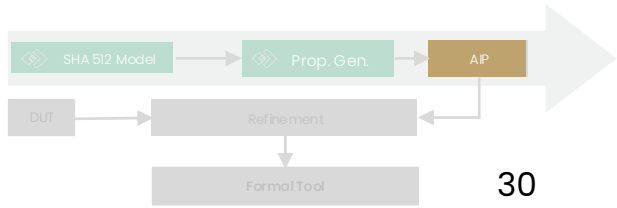
Simulation

Property Generation

Properties Review

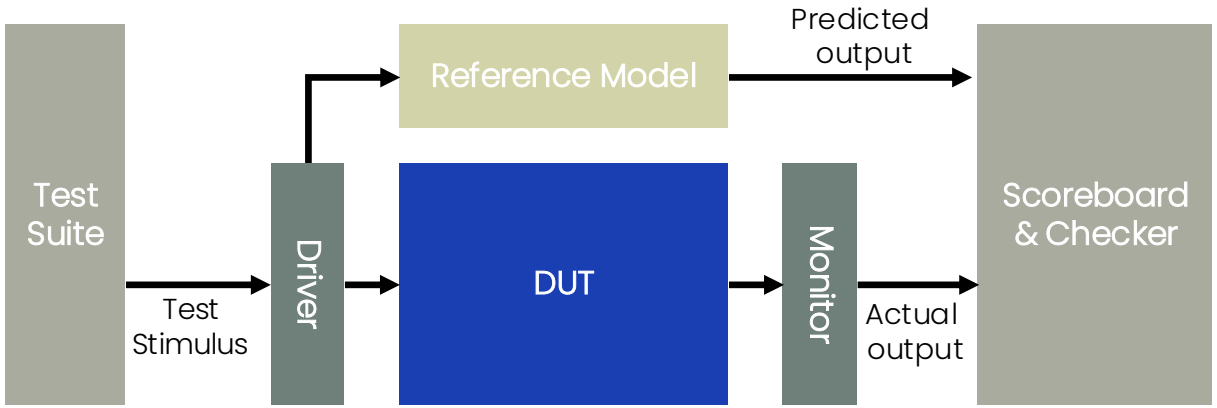
SHA512 State Machine



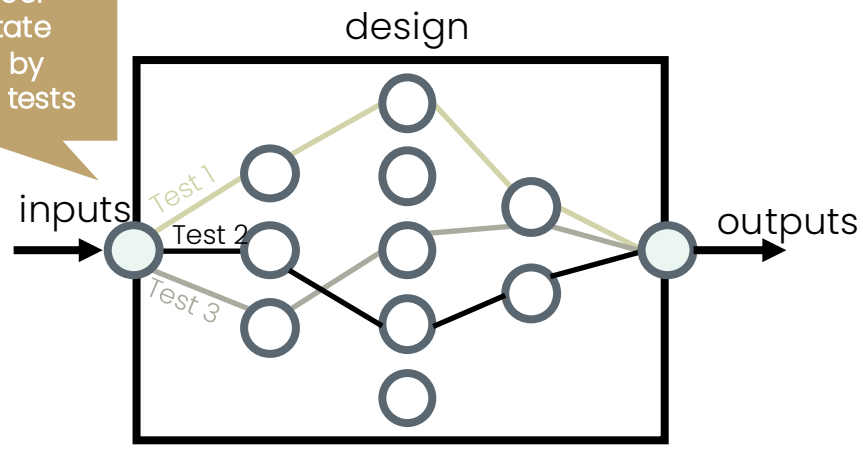


Model Refinement

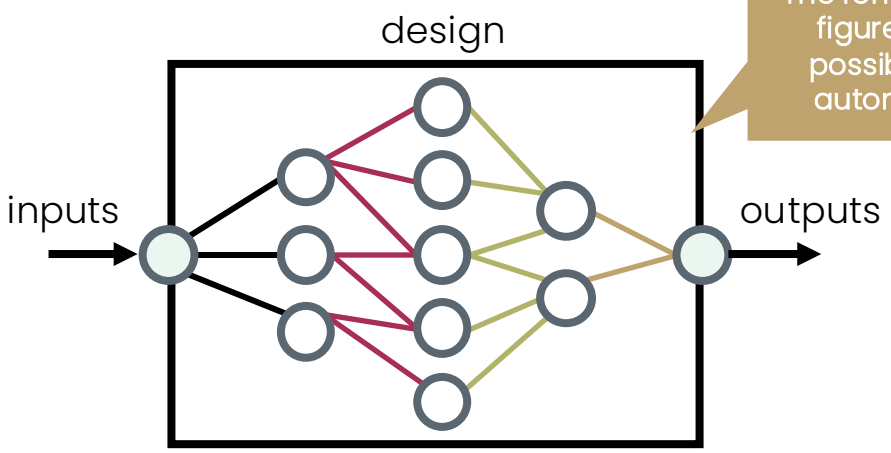
Simulation VS Formal

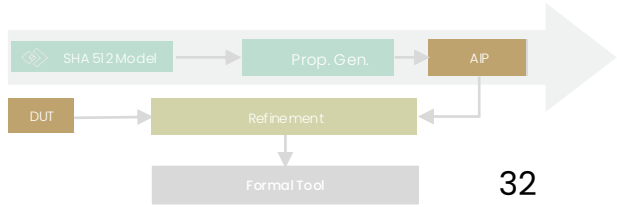


The engineer ensures state coverage by formulating tests



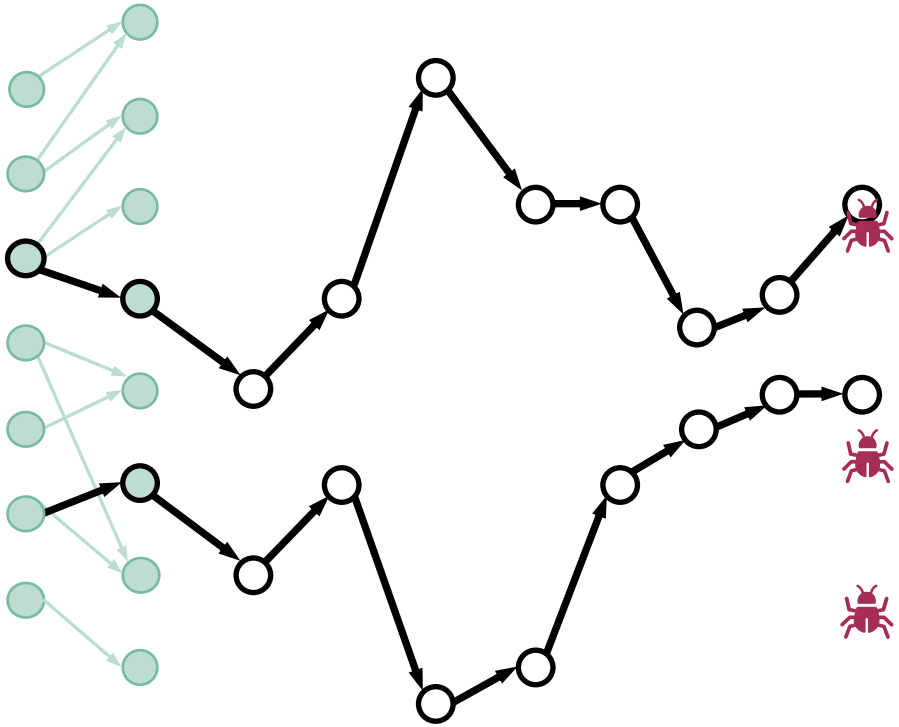
The formal engine figures out all possible paths automatically



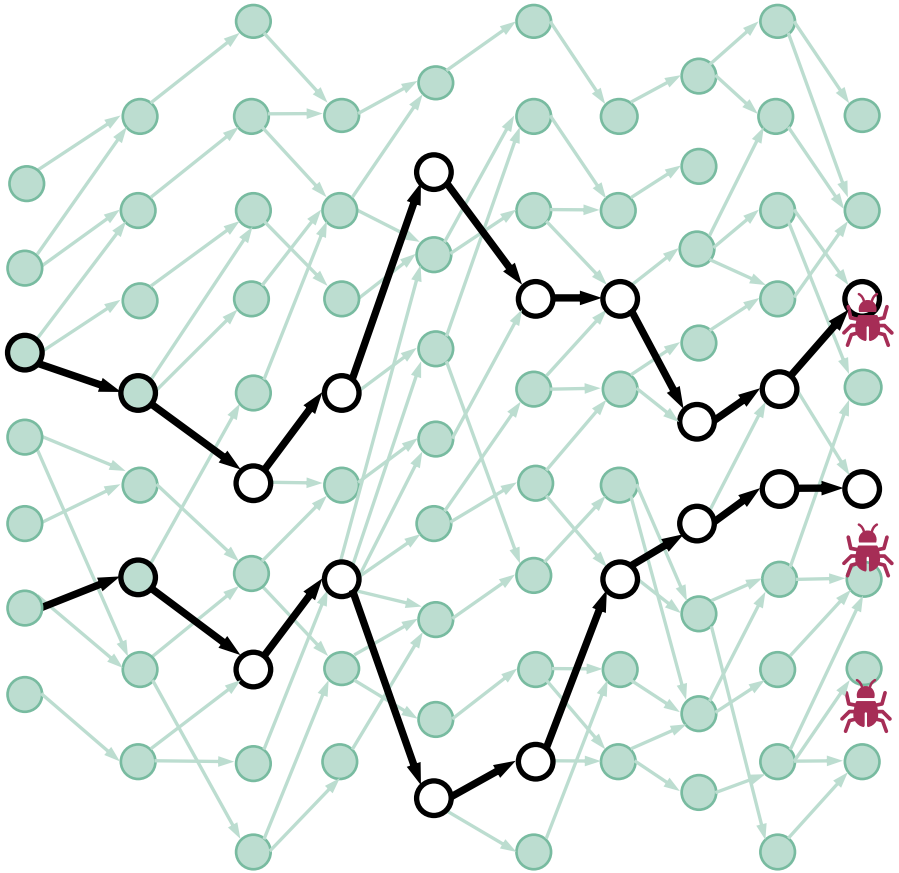


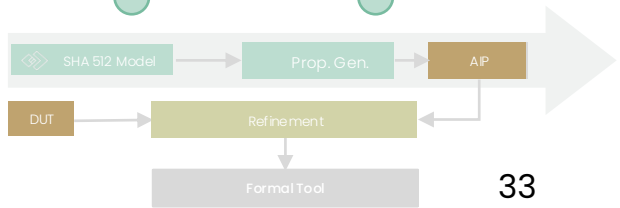
Simulation VS Formal

UVM



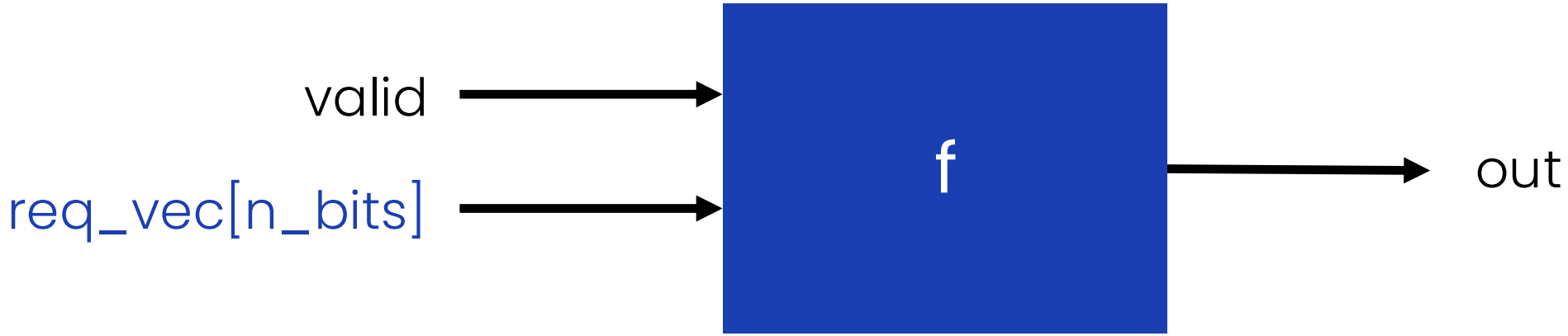
Formal



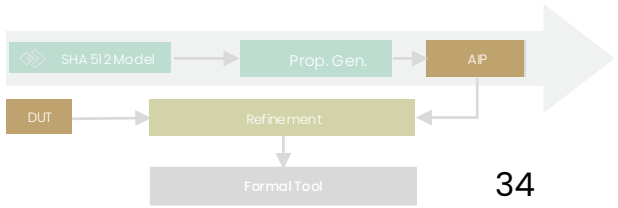


Practical Example

To get a feeling for exponential blow up



out = valid && (req[0]|| req[1] … || req[n-2]) && req[n-1]



Simulation VS Formal

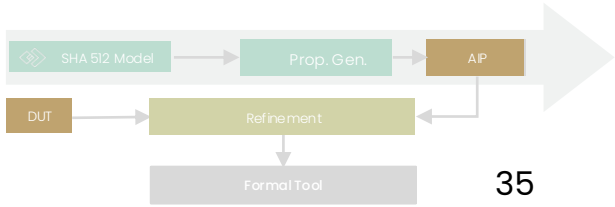
n = 100

Simulation *(if one run takes 10^{-9} seconds (nano))*

about 40 trillion years, ~120 trillion football seasons

Formal

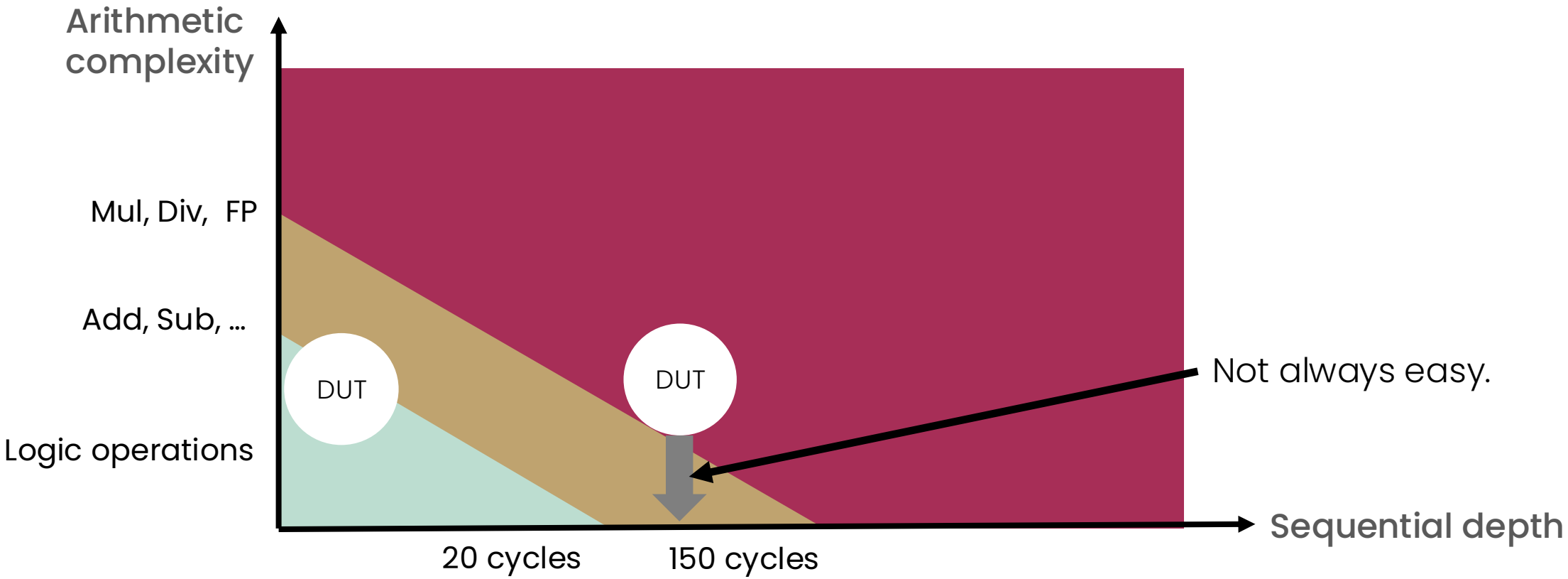
Less then 10 min

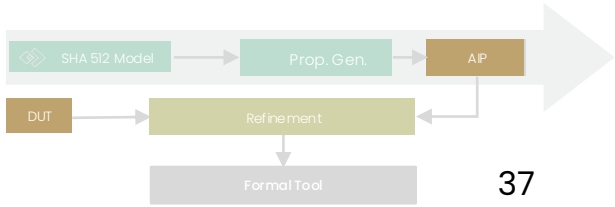


Property Checking

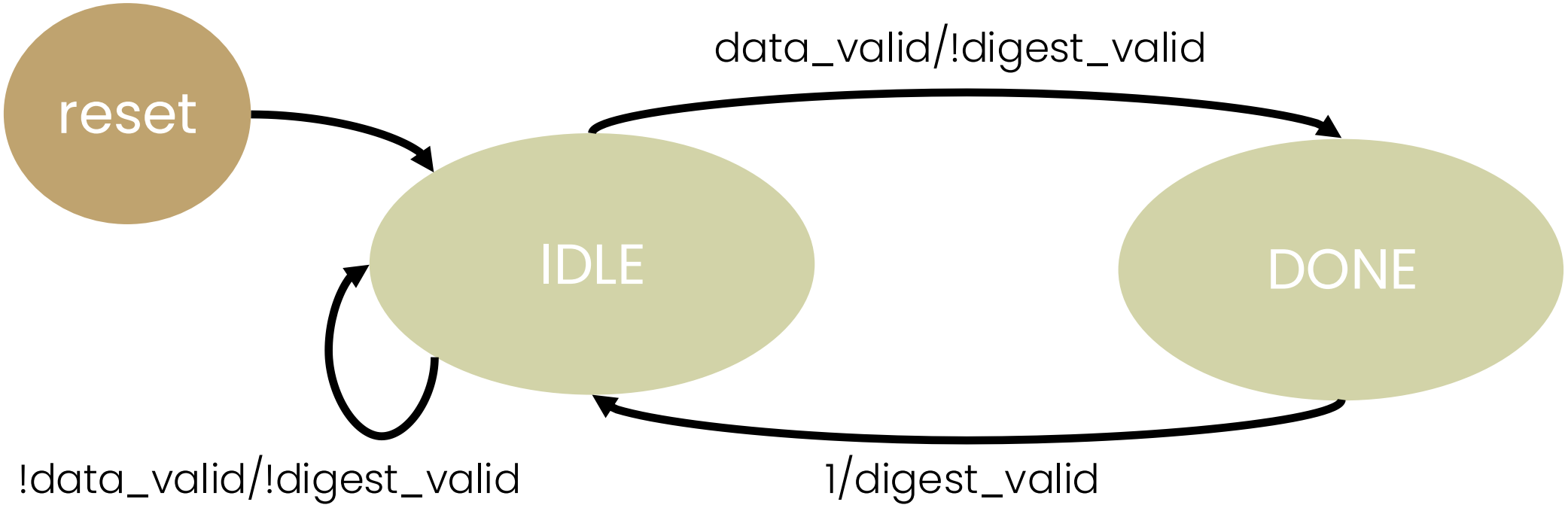
Proof Complexity

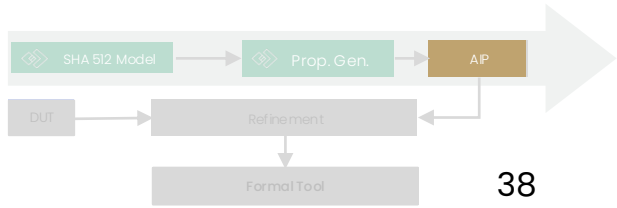
What can we do about non-converging proofs?



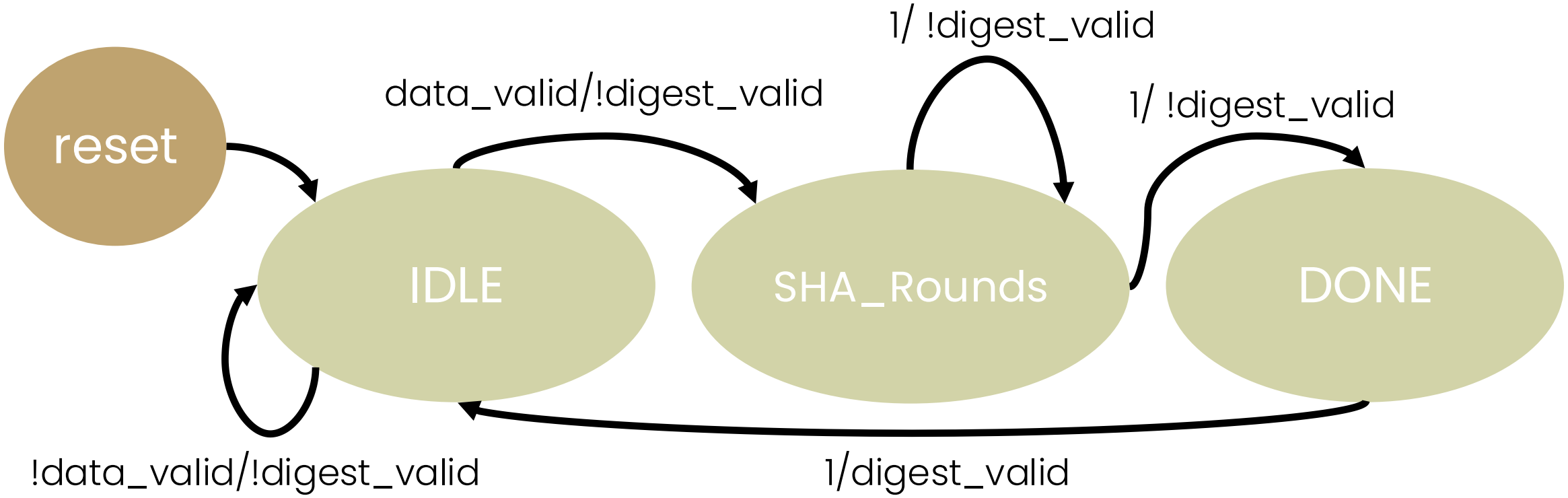


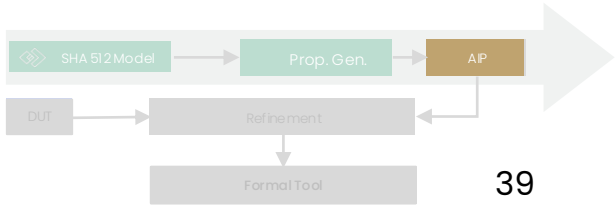
SHA512 State Machine





SHA512 State Machine





Reducing Complexity

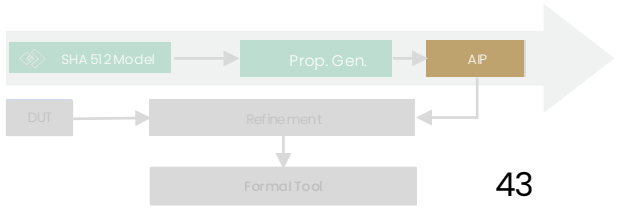
Simulation

Property Generation

Model to Property

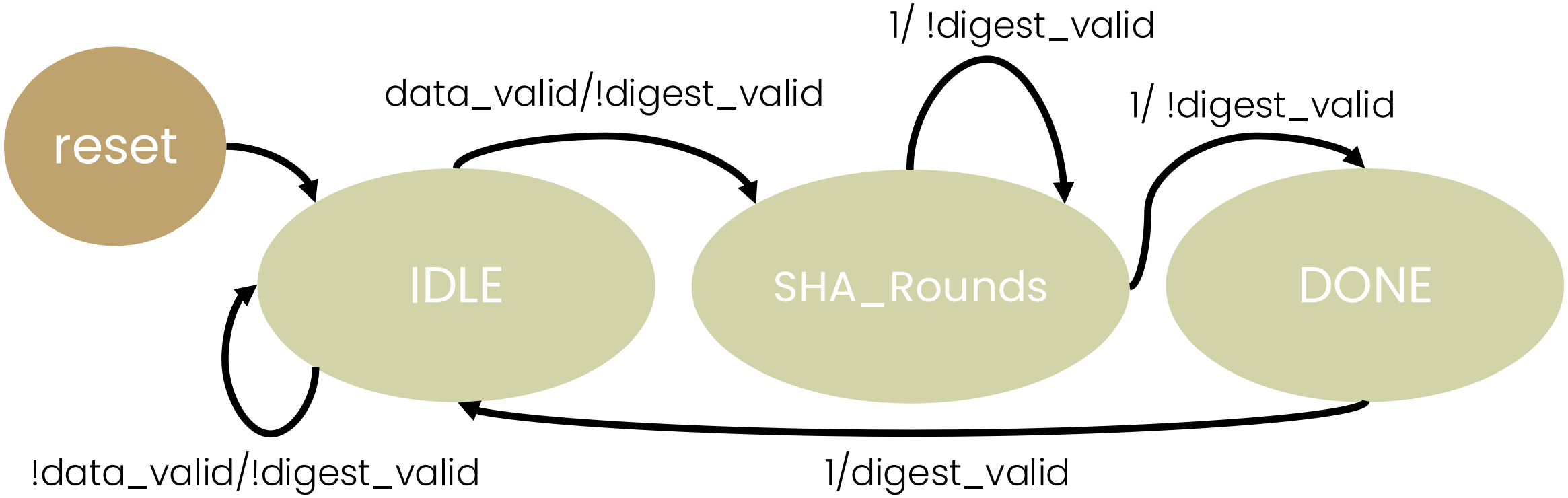
```
1. SHA_Input->read(SHA_in, “IDLE”);
2.
3. if(init){
4.       // Some Computation
5. }
6. // Some Computations
7. for(i = 0; i < NUM_ROUNDS; ++i){
8.   insert_state(“SHA_Rounds”);
9.   if(i < 16){
10.    // Some Computation
11.  } else {
12.      // Some Computation
13.  }
14.  // Some Computation
15. }
16. insert_state(“DONE”);
```

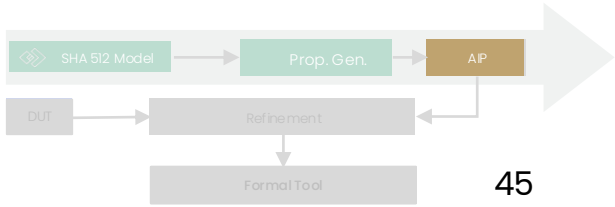
```
1. property fsm_SHA_Rounds_to_SHA_Rounds_1_p;
2.         SHA_Rounds &&
3.         i >= 16 &&
4.         ((i + 1) < 80)
5. |->
6.         ##1 ((SHA_Input_rdy == 0) &&
7.                 (out_vld == 0)) and
8.         ##1
9.         SHA_Rounds &&
10.        $stable(H) &&
11.        W == $past(W_2_i, 1) &&
12.        a == ($past(T1_1_i, 1) + $past(T2_0_i, 1)) &&
13.        b == $past(a, 1) &&
14.        c == $past(b, 1) &&
15.        d == $past(c, 1) &&
16.        e == ($past(d, 1) + $past(T1_1_i, 1)) &&
17.        f == $past(e, 1) &&
18.        g == $past(f, 1) &&
19.        h == $past(g, 1) &&
20.        i == (7'd1 + $past(i, 1));
21. endproperty
```



Properties Review

SHA512 State Machine



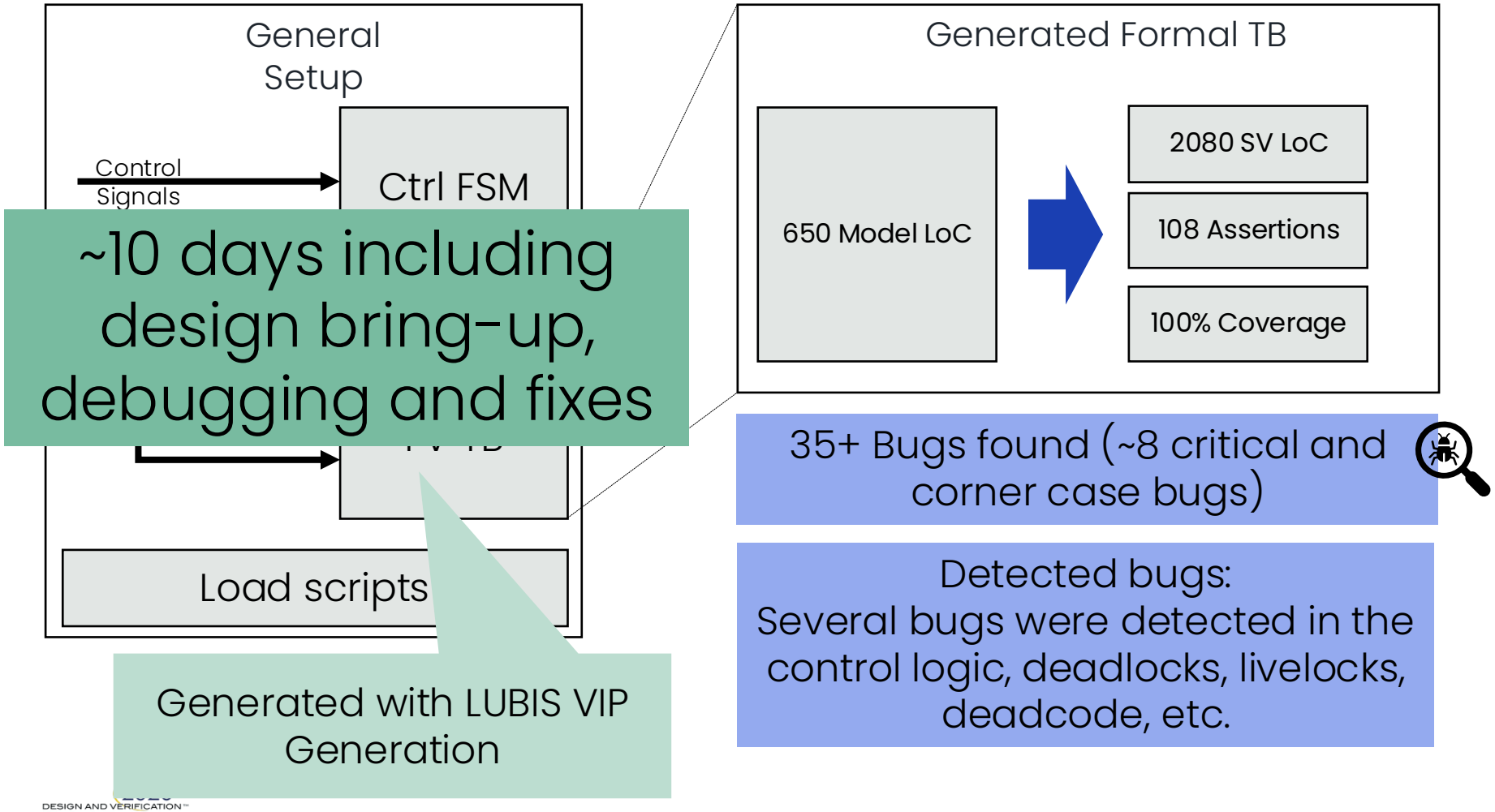


Model Refinement

Property Checking

One More Success Story

Control FSM with 30+ states - TOP3 US SEMI



~10 days including design bring-up, debugging and fixes

35+ Bugs found (~8 critical and corner case bugs)

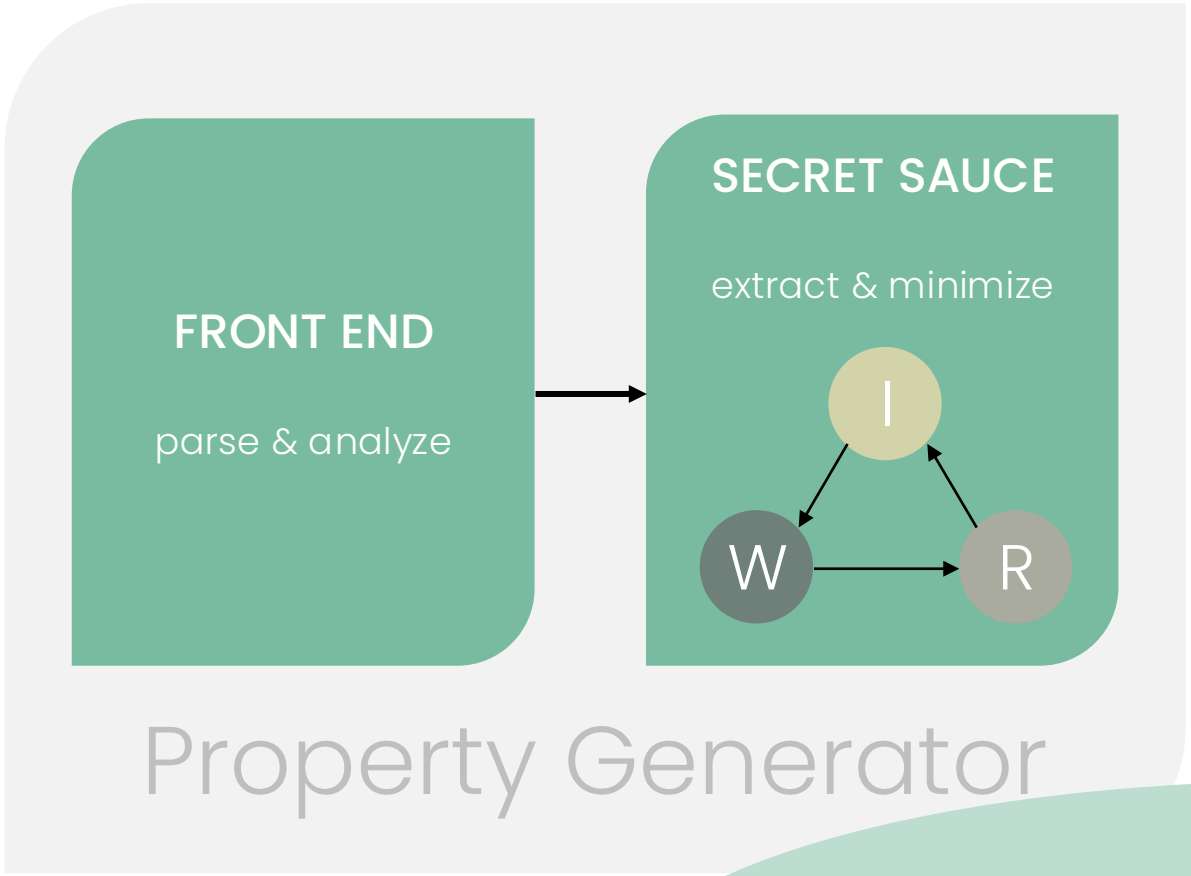
Detected bugs:
Several bugs were detected in the control logic, deadlocks, livelocks, deadcode, etc.

Generated with LUBIS VIP Generation

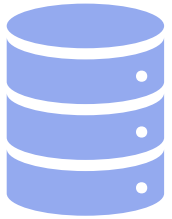
Property Generator

From model to verification IP in minutes

Abstract Model



Formal Verification IP

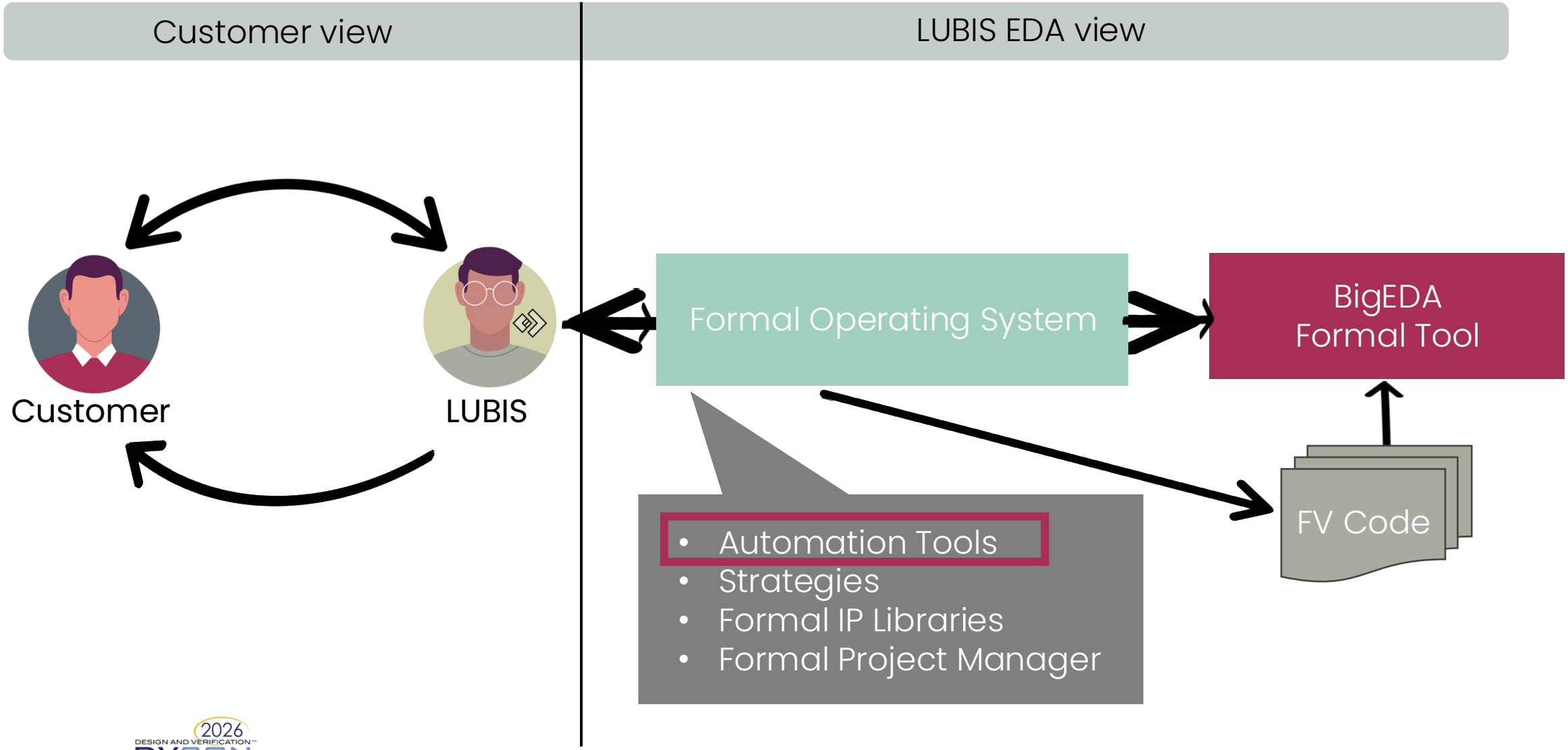


- Easy to read SVA properties
- Properties check design intend
- Full coverage of functional behaviour

Let's wrap this up

With property generator we are 50% faster compared to manual effort and our properties are 100% correct.

From good old consulting



Want to learn more?

Follow us to get notified on news about Formal Verification solutions

Tobias Ludwig

CEO

Email: Tobias.ludwig@lubis-eda.com

Web: www.lubis-eda.com

LinkedIn