



LUBISEDA

**Property Generator: Generation of
Formal Assertion IP**

Tobias Ludwig, CEO

Gen AI and Coding Tasks



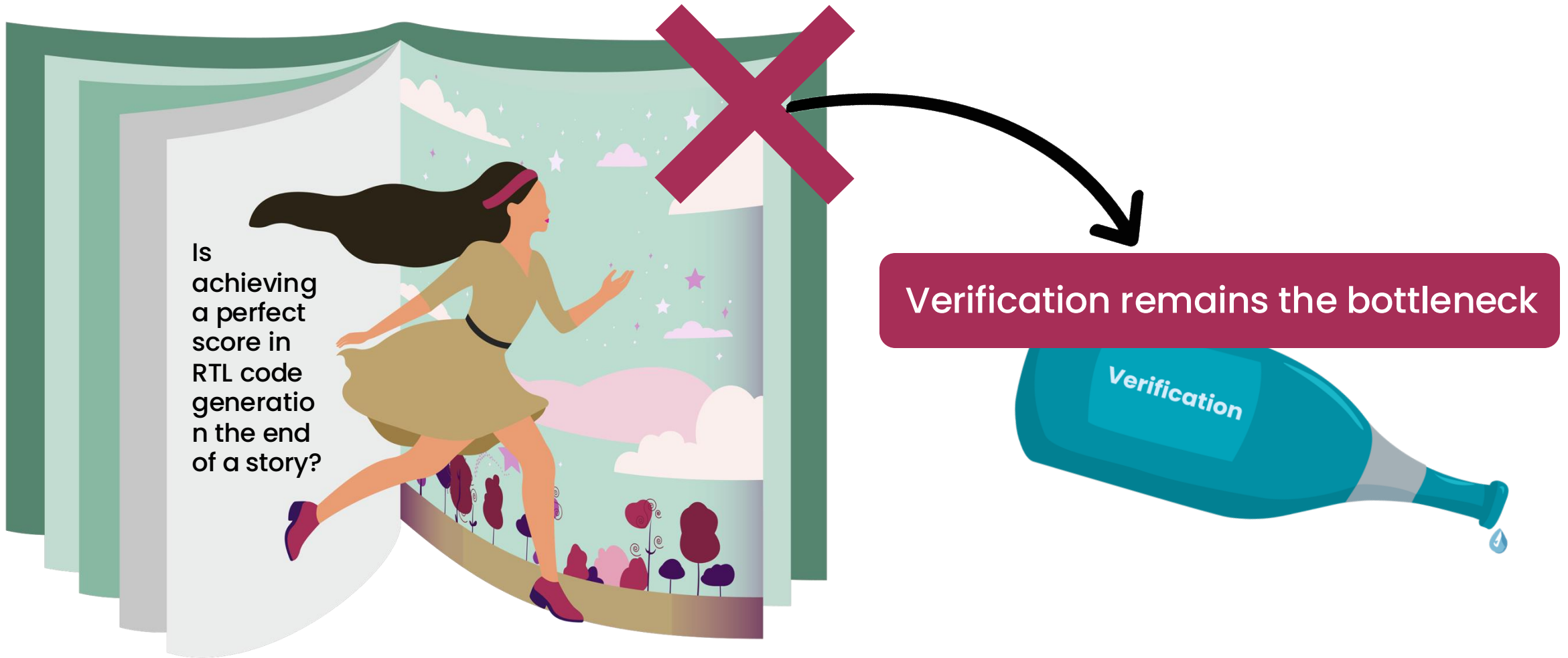
Software domain

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

Not true for RTL code generation ... yet

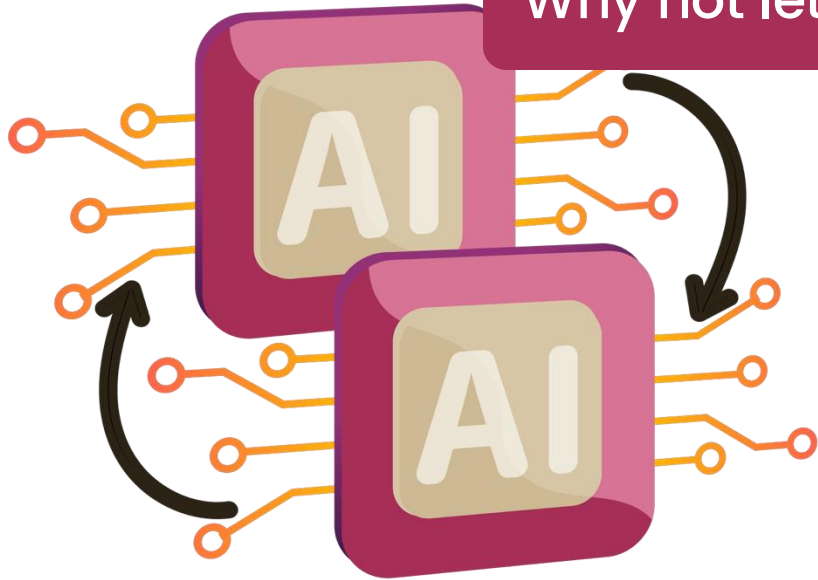
- Massive progress on VerilogEval benchmark, from 26% GPT3.5 to 63% GPT4o to 98% Agentic systems
- CVDP currently hovers around 30%

Gen AI and RTL Design



Gen AI and RTL Design Verification

Why not let AI verify AI?



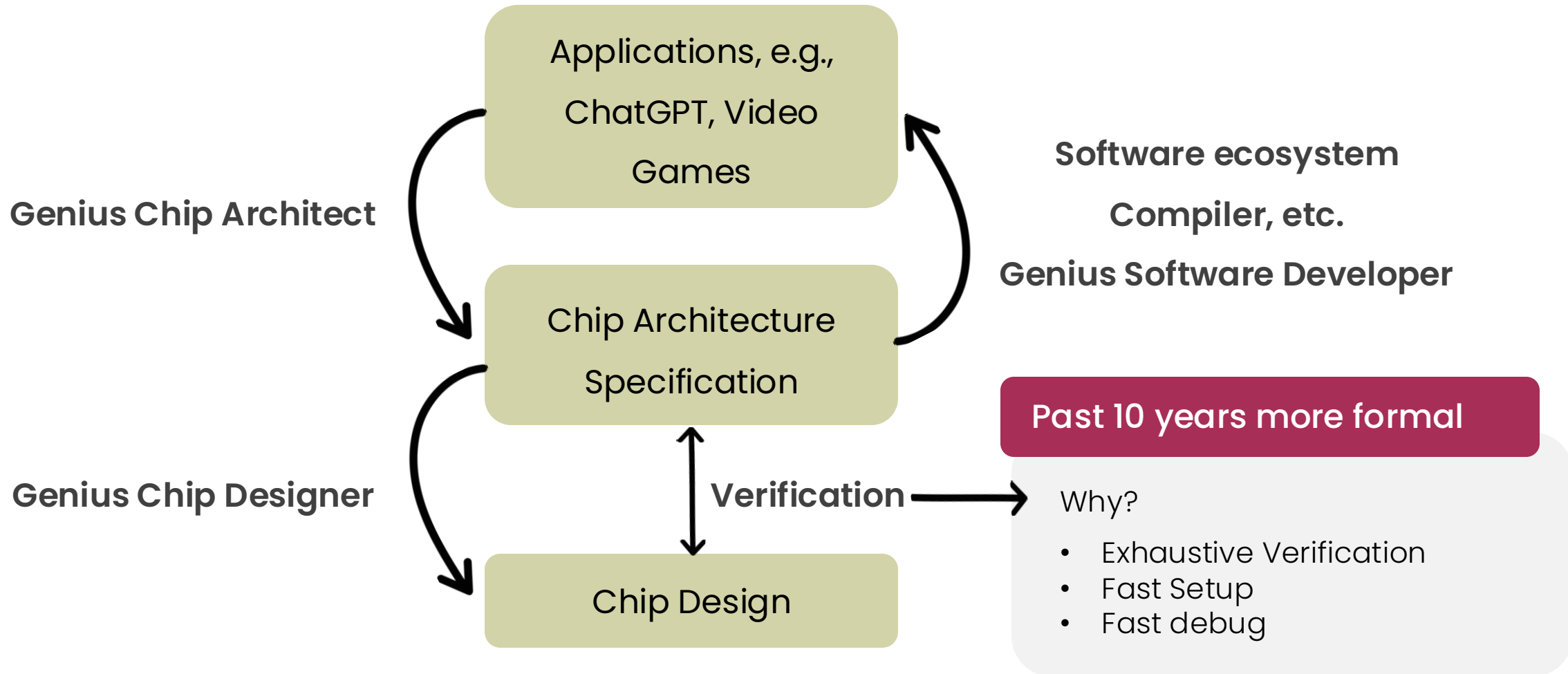
Will not scale for chip design ... why?

- 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

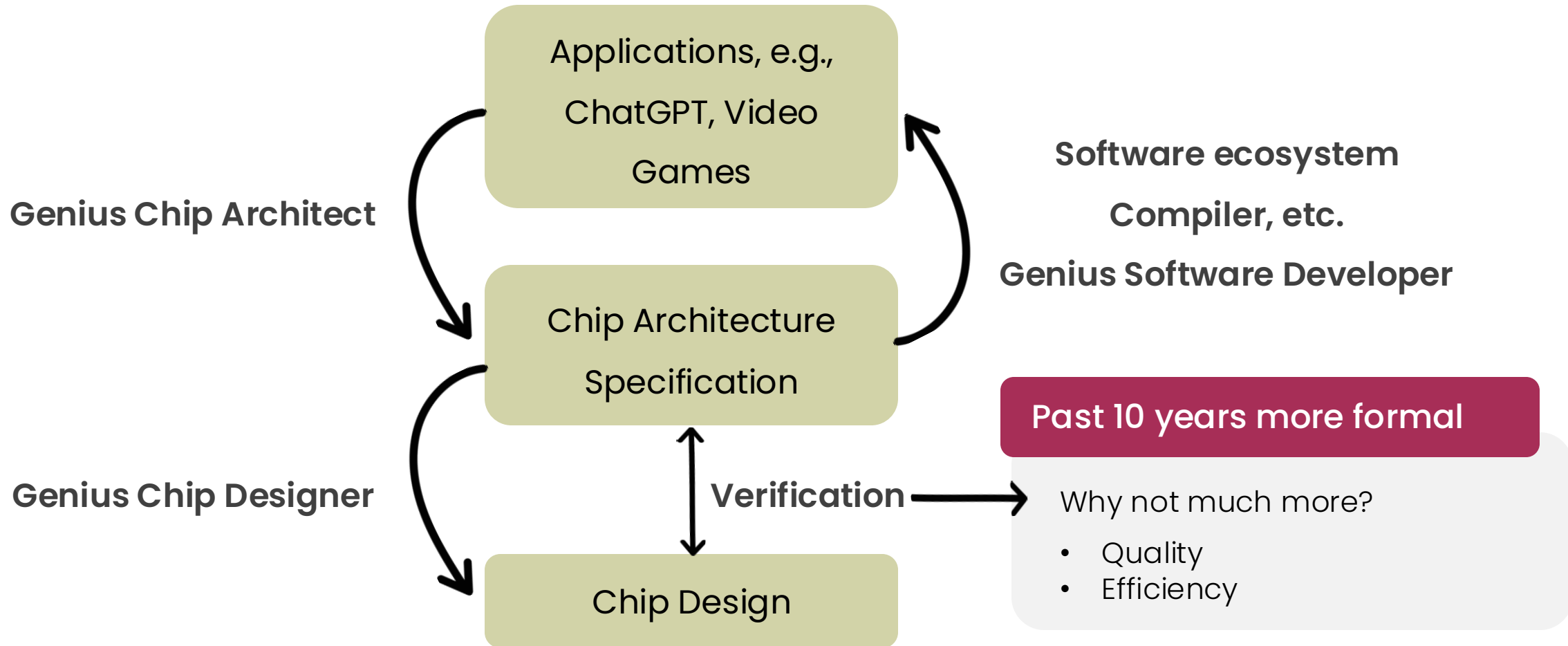
What it takes to have a successful chip?

Tens of thousands of engineer days



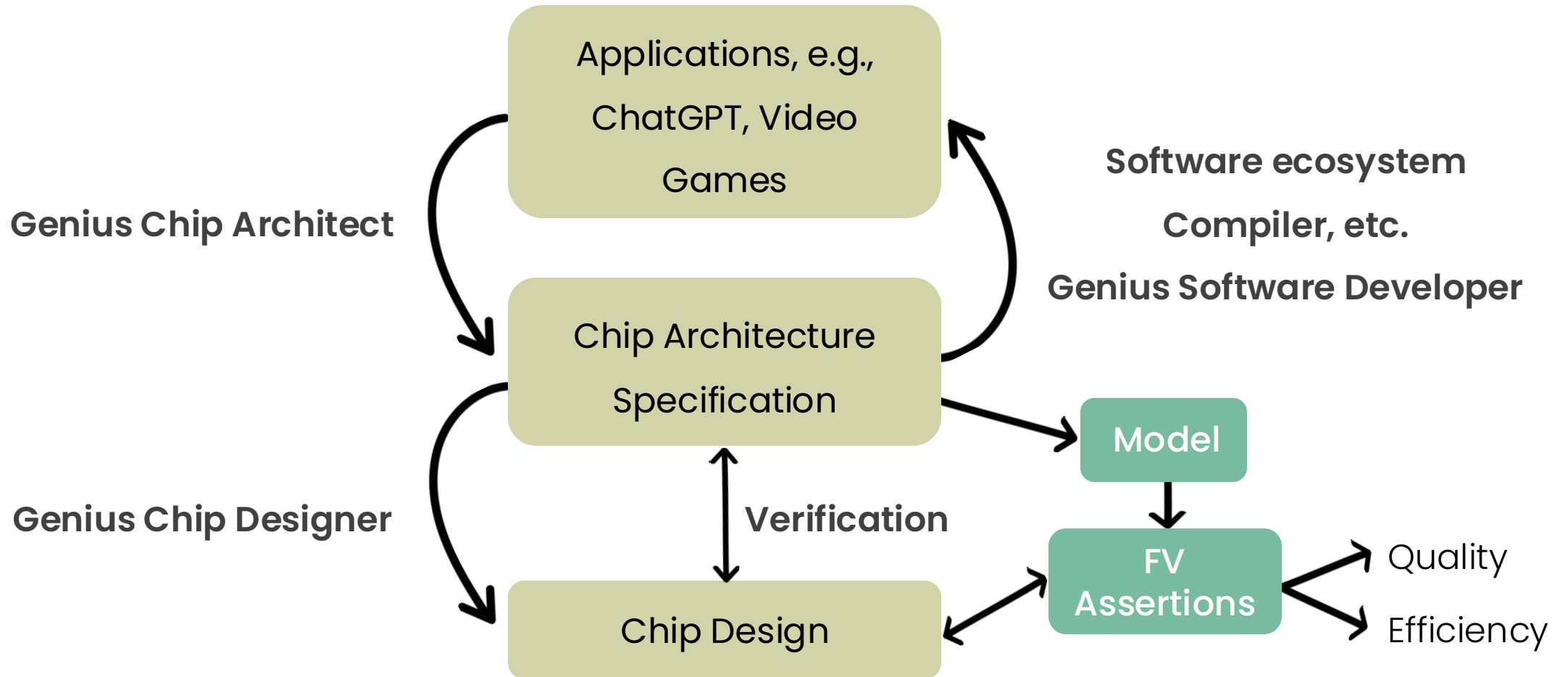
What it takes to have a successful chip?

Tens of thousands of engineer days



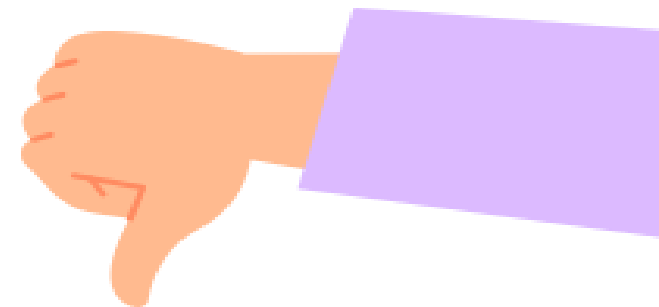
What it takes to have a successful chip?

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. );
```

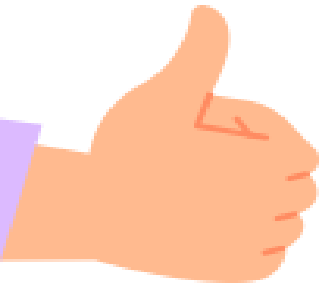
Turning intend into properties

```
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. }
```

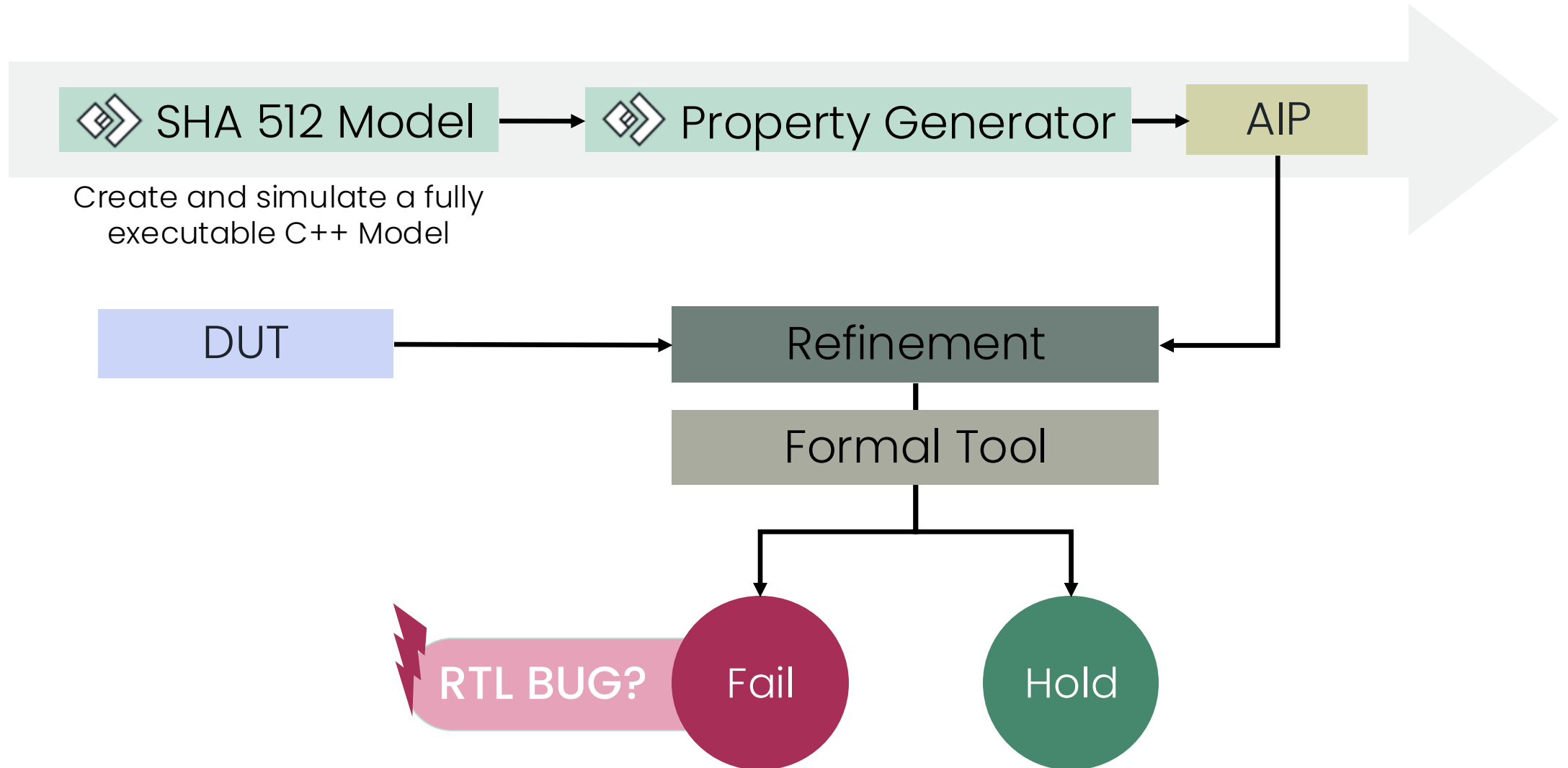
```
1. // FIFO watermark comparator context
2. // fill_level: current FIFO occupancy (0..DEPTH)
3. // watermark: programmable threshold
4. // above_wm: 1 when fill_level > watermark
5. // below_wm: 1 when fill_level <= watermark
6.
7. above_a: assert property (disable iff(!rst_n)
8.     (fill_level >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



**No free
lunch**



Here is what we are going to do



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. }
```

Model to Property

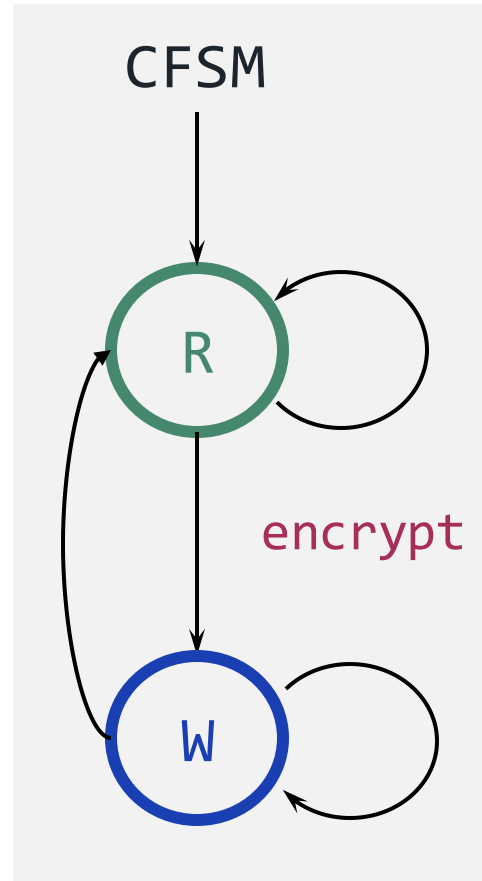
```
1. SHA_Input->read(SHA_in, "IDLE");
2.
3. if(init){
4.     // Some Computation
5. }
6. // Some Computations
7. for(i = 0; i < NUM_ROUNDS; ++i){
8.     insert_state("SHA_Rounds");
9.     if(i < 16){
10.        // Some Computation
11.    } else {
12.        // Some Computation
13.    }
14.    // Some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

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  
        } } };
```

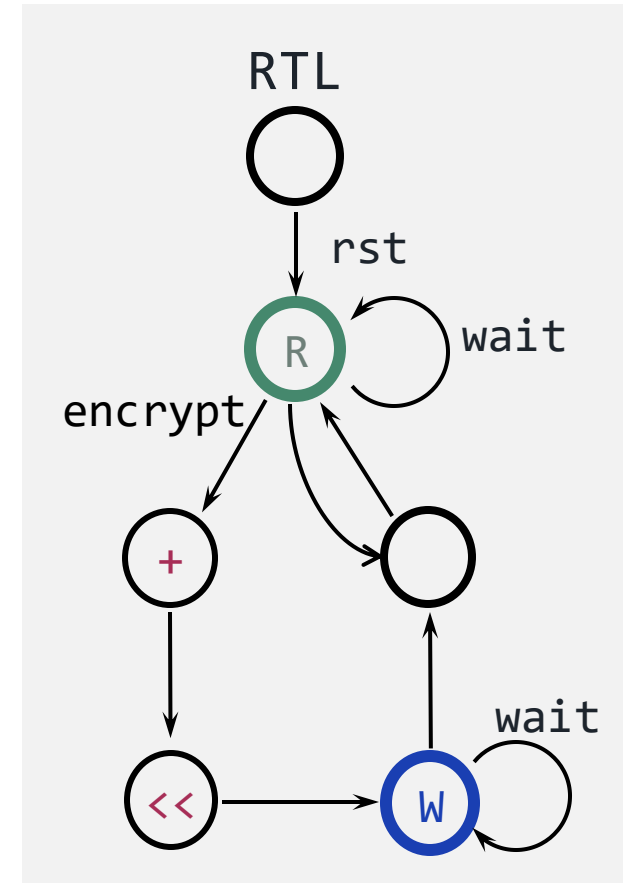
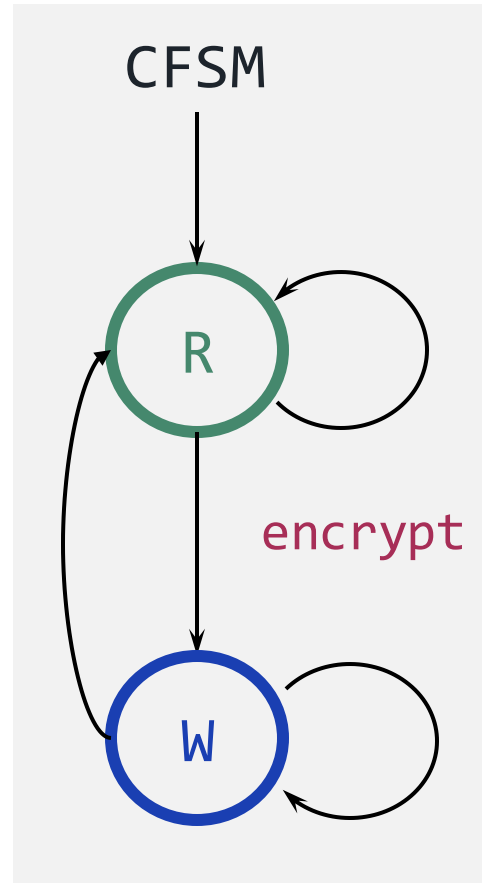


```
Property encrypt;  
assume:  
    at t: state == R;  
    at t: data_valid == 1;  
prove:  
    at t+n: out(enc(data@t));  
    at t+n: state == W;  
endproperty;
```

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

```
1: property encrypt(length);
2:   //local variables
3:   logic[1023:0] data_0;
4:   // store values @t
5:   ##0 (valid_data, data_0 = bus_in_data()) and
6:   // triggers
7:   ##0 state() == R
8: |->
9:   ##length state() == W and
10:   ##length bus_out()==enc(data_0)
11: endproperty;
```

```
1. function logic[1023:0] bus_in_data()
2.   return data_in;
3. endfunction;
```

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

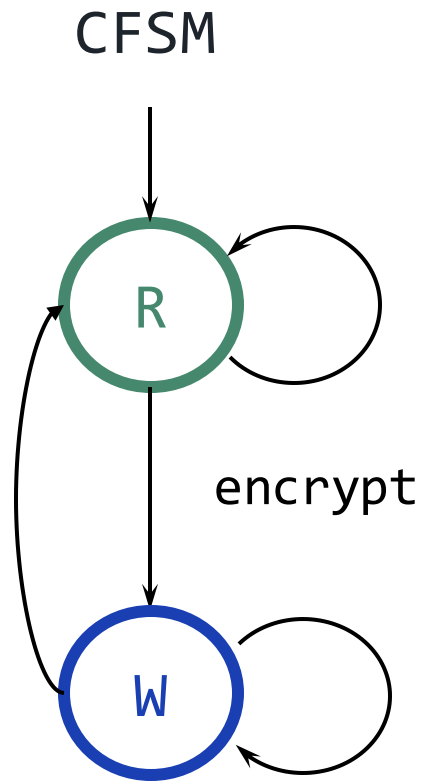
Refining the model

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

```
1: property encrypt(length);
2:   //local variables
3:   logic[1023:0] data_0;
4:   // store values @t
5:   ##0 (valid_data, data_0 = bus_in_data()) and
6:   // triggers
7:   ##0 state() == R
8: |->
9:   ##length state() == W and
10:  ##length bus_out()==enc(data_0)
11: endproperty;
```

```
1. function logic[1:0] state()
2.   if(state_reg == 2'b00)
3.     return R
4.   else if(state_reg == 2'b01)
5.     return W
6. endfunction;
```

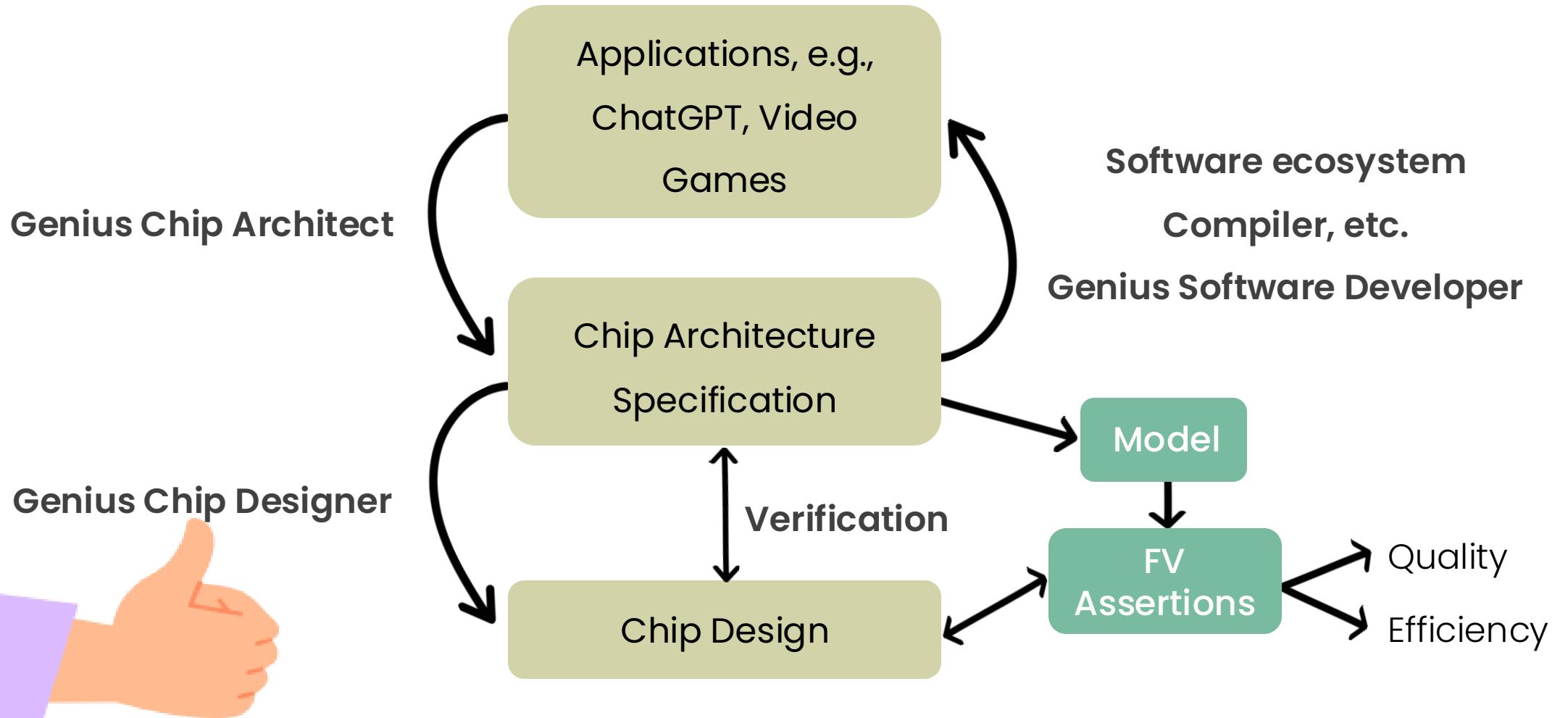
Generating properties without gaps



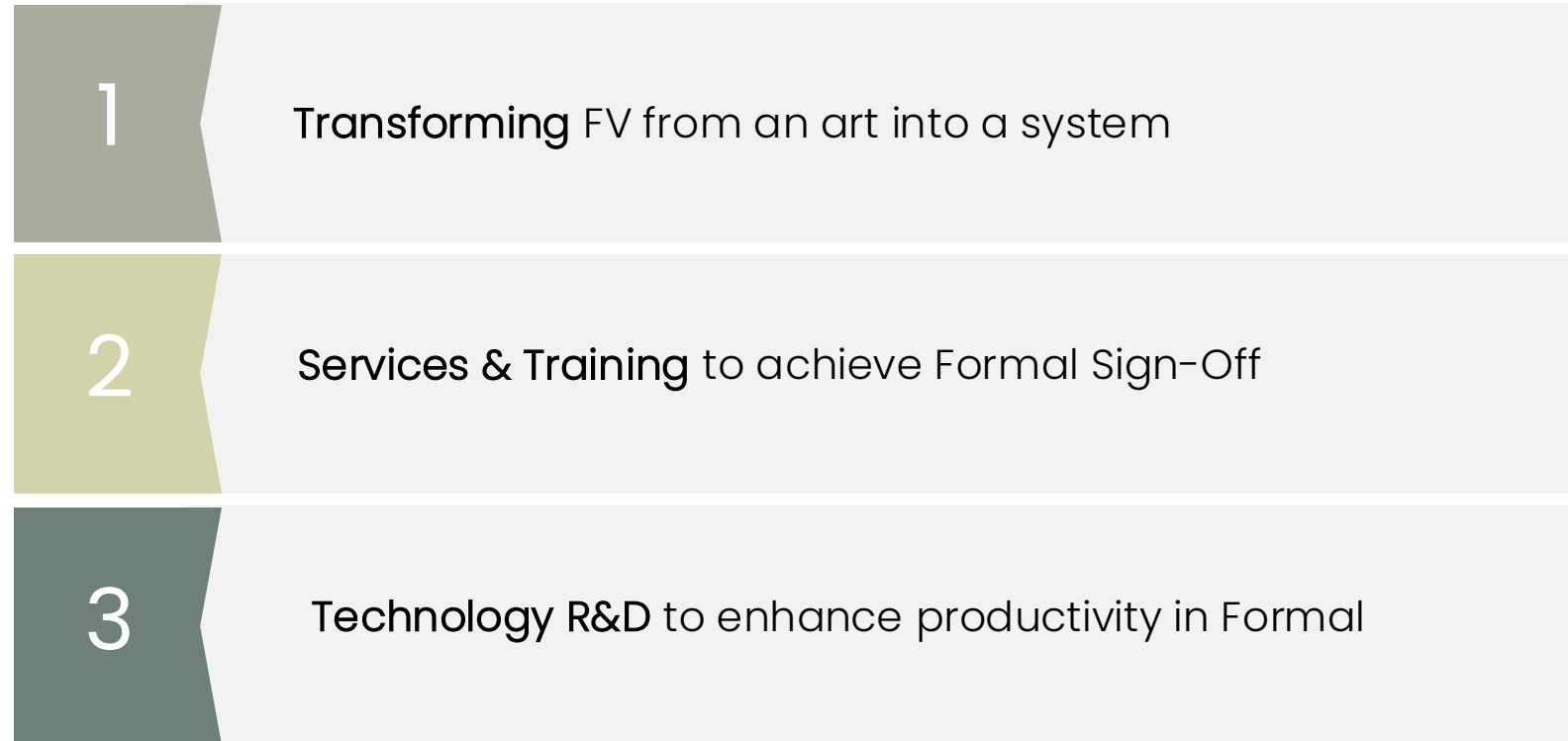
FSM Path	Property in Assertion IP
Reset -> Read	reset_p
Read -> Read	read_to_read_p
Read -> Write	read_to_write_p
Write -> Write	write_to_write_p
Read -> Read	write_to_read_p

What it takes to have a successful chip?

Tens of thousands of engineer days



We help chip teams reach formal sign-off



Team
35+



Kaiserslautern,
Germany



We love formal



since 2020

What we've delivered

250+

Projects completed

Comprehensive coverage across CPU, cache, and interconnect projects

6

Fortune 100 Customers

Trusted by the world's leading semiconductor companies

750+

Design Bugs Uncovered

Critical issues identified and reported through formal verification



Want to learn more?

Follow us to get notified on news about: Formal services, training and software



Tobias Ludwig

CEO

 Email: Tobias.ludwig@lubis-eda.com

 Web: www.lubis-eda.com

LinkedIn

