# Uncover
## Functional Coverage Made Easy

Akash S

Rahul Jain

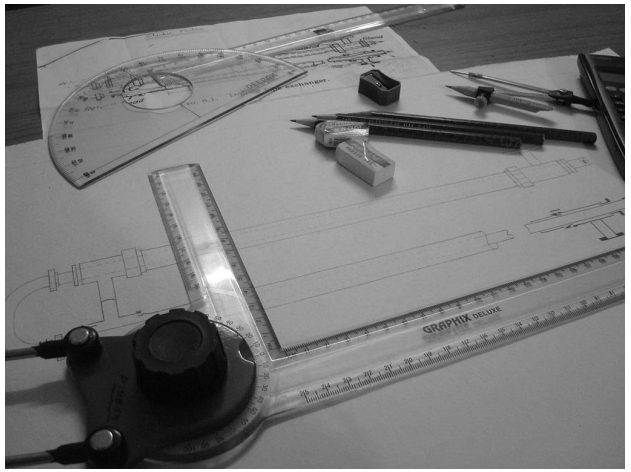
Gaurav Agarwal

NVIDIA

accellera SYSTEMS INITIATIVE™

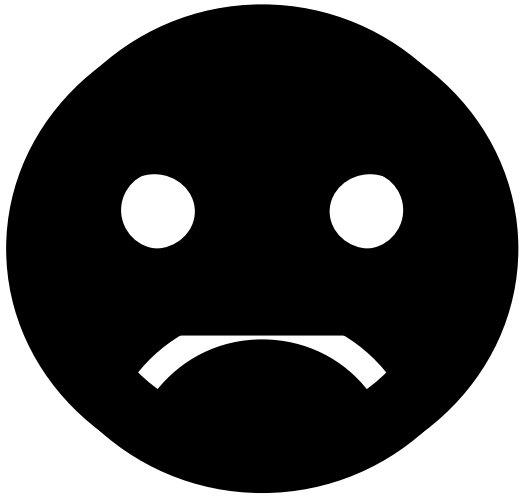
2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

## You too?



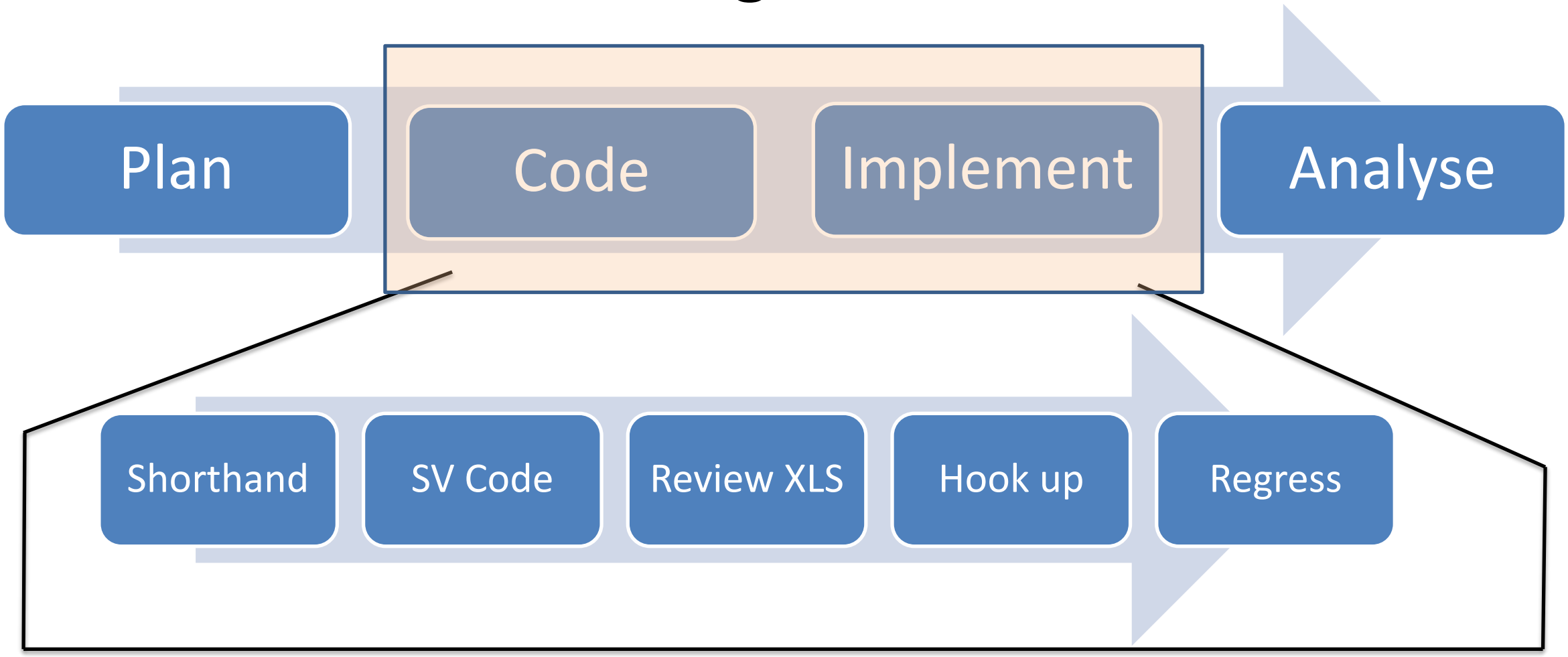

# What makes coverage coding tedious?

- **Monotony**
  - Bland work, not challenging enough!
- **Repetition**
  - Widespread across covergroups, projects.
- **Inconsistency**
  - Differences in structure, model, style

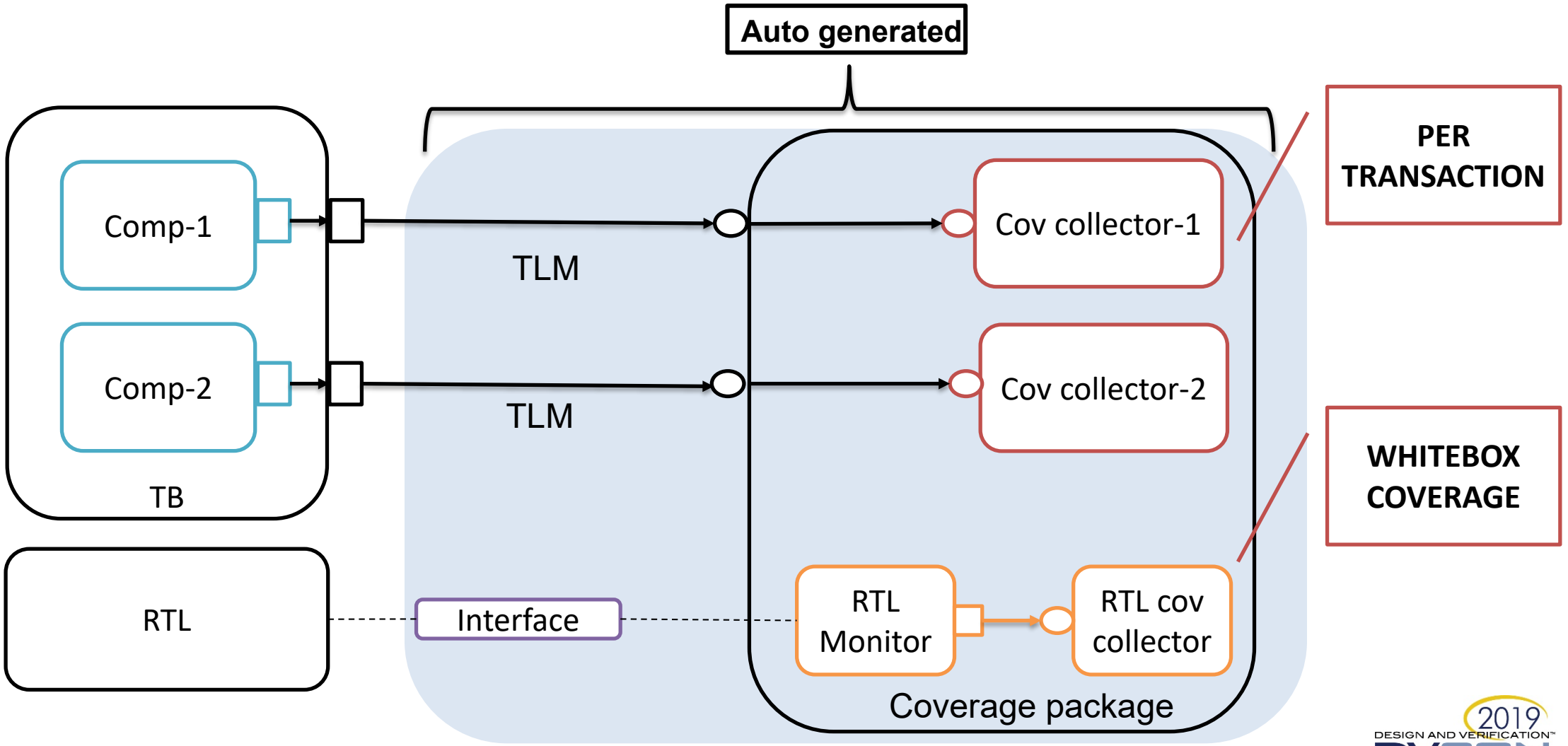


"Thou shalt keep verif engineers happy"

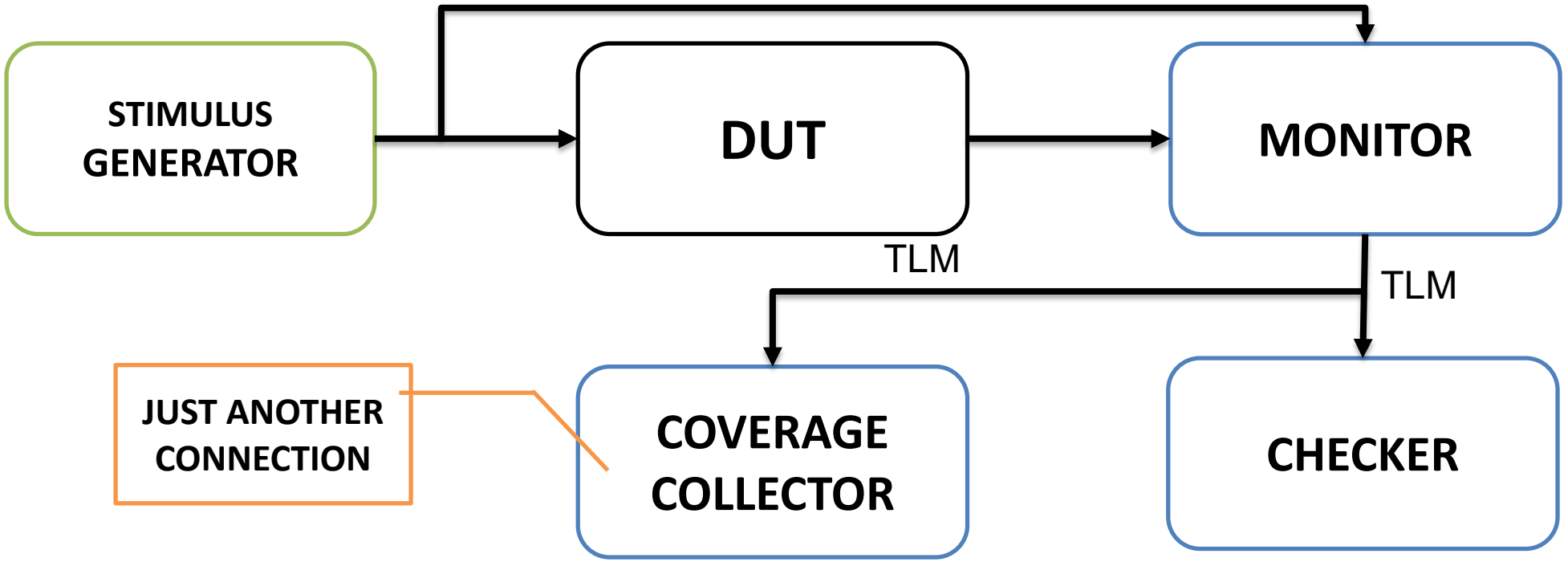
# The Big Picture



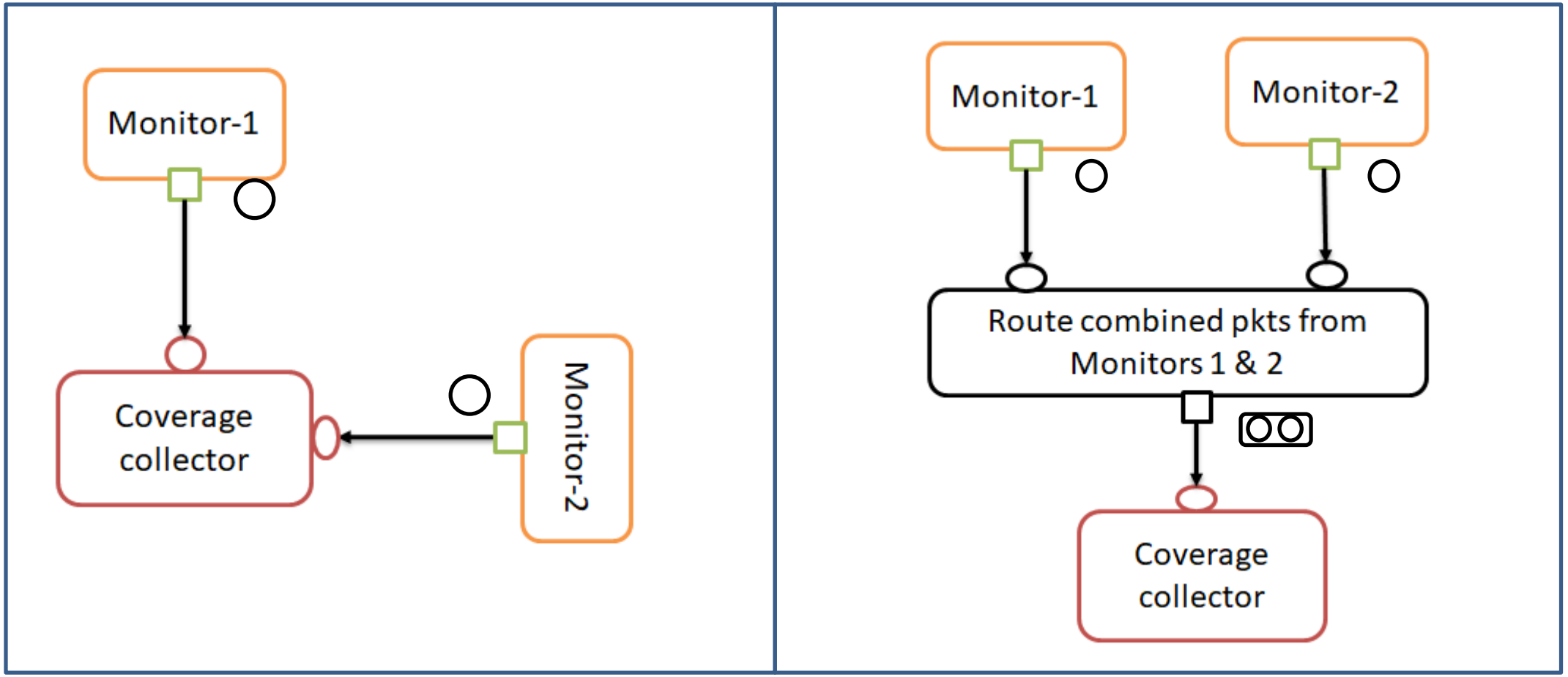
# “Thou shalt **automate the mundane**”



# “Thou shalt be **TRANSACTION-WISE**”

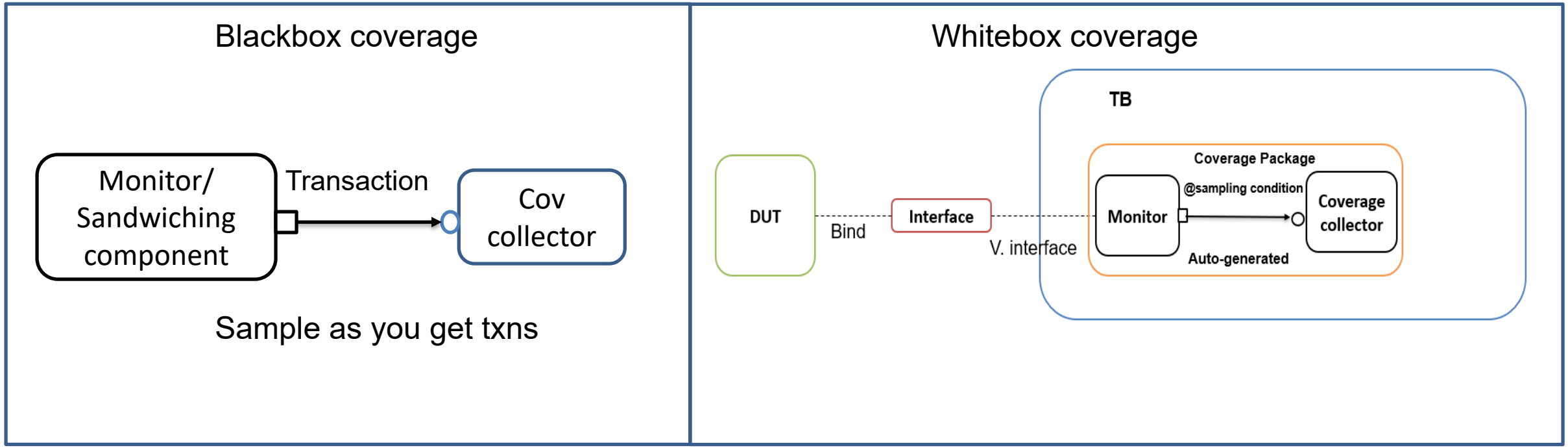


# “Thou shalt **separate concerns**”



# “Thou shalt be **flexible**”

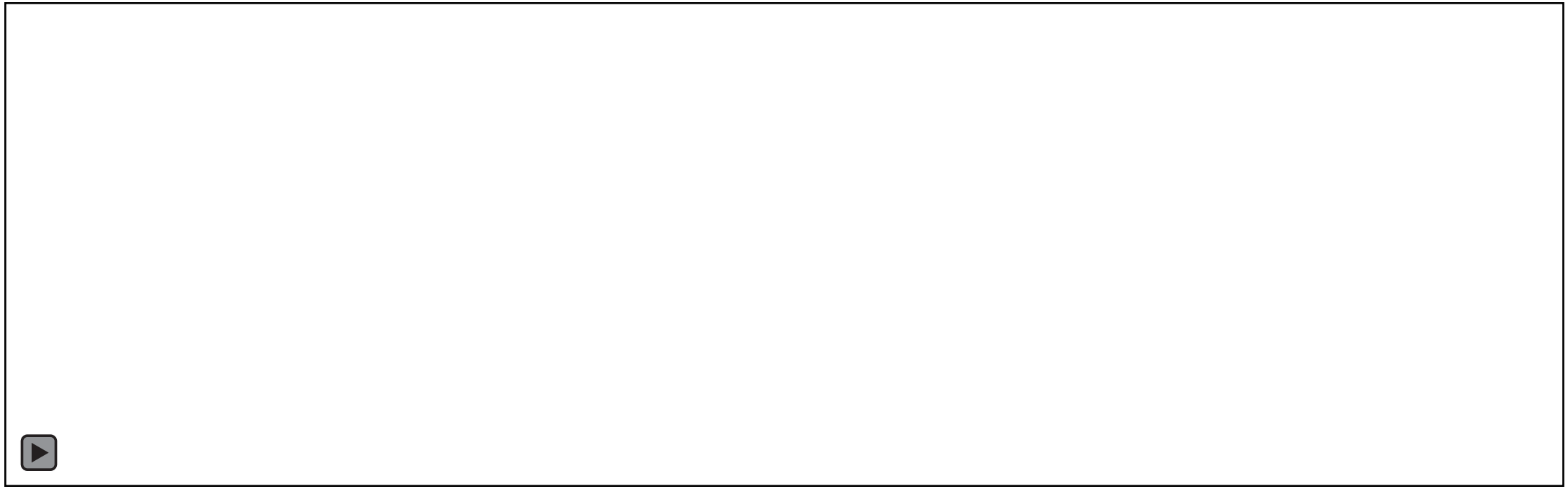
- Flow adaptable to whitebox coverage.
- Use the same infra with additional command line arguments



# “Thou shalt be **concise**”

## Reviewability

- Structured XLS for review of implementation.
- Only coverage model, cut the rest.



# “Thou shalt be **concise**”

## Coding: The Shorthand Notation

- One-line CG/CP/cross definitions
- Auto-generate whatever possible
- No repetitions
- Leverage on patterns
- Customizable sequence coverage

## Generated File

```
1  //Generated by Auto Cov flow. Do not edit!
2
3  import uvm_pkg::*;
4
5  class cg_basic_txn_cov extends uvm_subscriber#(cg_basic_txn) ;
6      `uvm_component_utils(cg_basic_txn_cov)
7      cg_basic_txn obj;
8
9      covergroup cg_basic_txn_1;
10         cp_addr :coverpoint  obj.addr  {
11
12             bins ram = {[0:5]};
13             ignore_bins reserved = {[13:15]};
14             illegal_bins rom = {[6:12]};
15         }
16         cp_w_rn :coverpoint  obj.w_rn  {
17         }
18     endgroup: cg_basic_txn_1
19
20     covergroup bus_txn;
21         cp_wr_data : coverpoint  obj.wr_data  iff ( obj.w_rn ){
22             option.auto_bin_max = 10;
23
24             bins min[2] = {0};
25             bins max[] = {127,126};
26         }
27         read : coverpoint  obj.rd_data  iff ( !obj.w_rn ){
28         }
29     endgroup: bus_txn
30
31     function new(string name, uvm_component p = null);
32         super.new(name,p);
33         bus_txn=new;
34         cg_basic_txn_1 = new;
35     endfunction: new
36
37     function void pre_sample();
38
39     endfunction: pre_sample
40
41     function void post_sample();
42
43     endfunction: post_sample
44
45     function void write(cg_basic_txn t);
46         `uvm_info(get_full_name(), "In COVERAGE class write method. About to sample coverage", UVM_NONE)
47         obj = t;
48
49         if( posedge  clk )
50             bus_txn.sample();
51
52         cg_basic_txn_1.sample();
53
54         `uvm_info(get_full_name(), "Done sampling. Exiting COVERAGE class write method", UVM_NONE)
55     endfunction: write
56
57 endclass: cg_basic_txn_cov
```

**"The mundane"**

## Shorthand

```
1  siglist:
2      bit [3:0] addr;
3      bit [7:0] wr_data;
4      bit w_rn;
5      bit [7:0] rd_data;
6
7  cg_list:
8      cg : [at_least=1, per_instance = 1];
9          cp cp_addr: {addr} ram([0:5]), -rom([6:12]), *reserved([13:15]);
10         cp: {w_rn};
11     cg bus_txn: (posedge clk);
12         cp : {wr_data}(w_rn)[auto_bin_max = 10] min[2](0), max[](127,126);
13         cp read : {rd data}(!w rn);
```

# Shorthand: A Glance

**SV snippet**

```
//Comment for CG bus_txn.
covergroup bus_txn;
```

```
if( test_condition )
    bus_txn.sample();
```

**(Inside write function of cov collector)**

```
cp_addr :coverpoint  obj.addr  {

    bins ram = {[0:5]};
    ignore_bins reserved = {[13:15]};
    illegal_bins rom = {[6:12]};

}
```

**Shorthand snippet**

```
cg bus_txn: {test_condition};
//Comment for CG bus_txn
```

```
cp cp_addr: {addr} ram([0:5]), -rom([6:12]), *reserved([13:15]);
```

## SV snippet

```
cp_wr_data : coverpoint  obj.wr_data  iff ( obj.w_rn ){
    option.auto_bin_max = 10;

    bins min[2] = {0};
    bins max[] = {127,126};
    wildcard bins test = {4'b11?};

}
```

```
//Comment for cross first_attempt.
first_attempt  :cross wr_data,w_rn {
    option.goal = 100;

    bins bin_1 = binsof(cp_w_rn) intersect {1,2,3};

}
```

## Shorthand snippet

```
cp : {wr_data}{w_rn}[auto_bin_max = 10] min[2](0), max[](127,126), ?test(4'b11?);
```

```
cross first_attempt :{wr_data,w_rn}[goal = 100] bin_1(cp_w_rn{1,2,3});
//Comment for cross first_attempt
```

```
aliases_vars:
    bit write_en = test_alias_1 && test_alias_2;
    // Test var alias comment
    bit [5:0][4:0] write = xyz;
```

# “Thou shalt **not repeat** thyself”

- **Problem:**
  - CPs can’t be used across CGs
  - Define CP in every CG where needed.
  - 100s of repetitions!
- **Solution:**
  - One-point definition of all CPs
  - Command to copy CP/cross to a CG.
  - No repetitions!
- **Scenarios?**
  - speed x gears x sampling rate
  - speed x success; speed x error
  - speed x timeout

```
aliases_cp:
    cp cp_addr: {addr} ram([0:5]), -rom([6:12]), *reserved([13:15]);
    cross test : {addr,wr_data};
```

⇩

```
cp_copy cp_addr;
cross_copy test;
```

**(In CG)**

⇩

Replace above statements with actual shorthand definition before converting to SV

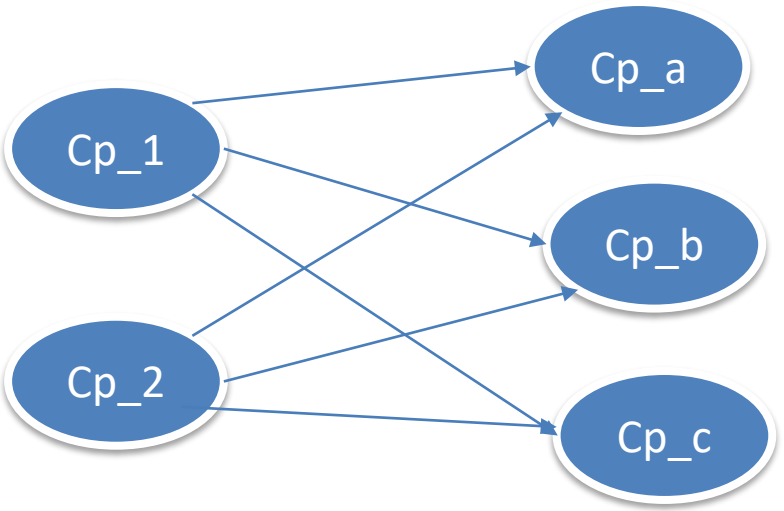
# “Harness patterns”

- Many-many relationships

```
cp_1Xcp_a : cross cp_1, cp_a;
cp_1Xcp_b : cross cp_1, cp_b;
cp_1Xcp_c : cross cp_1, cp_c;
cp_2Xcp_a : cross cp_2, cp_a;
cp_2Xcp_b : cross cp_2, cp_b;
cp_2Xcp_c : cross cp_2, cp_c;
```

→

```
^cross_expand("cp_1, cp_2", "cp_a, cp_b, cp_c");
```




- Irregular address ranges
- One-hot/one_cp_each
- Hop

# Macros: Gist

| Macro | Format | Example cfg syntax | O/P SV code |
|---|---|---|---|
| list | ^list(<vals>) | ^list{[2]{1}, *{2}, -{3}, {4}); | bins bin_list0[2] = {1};<br>ignore_bins ignore_list0 = {2};<br>illegal_bins illegal_list0 = {3};<br>bins bin_list1 = {4}; |
| hop | ^hop(min, max, step) | ^hop(1,5,2); | bins bins_hop_1 = { 1};<br>bins bins_hop_3 = {3};<br>bins bins_hop_5 = {5}; |
| interval | ^interval(h/d, min, max, break_1, break_2) | ^interval(d, 1, 10, 5,7);<br>(Similarly for hex) | bins bins_interval_1 = {[5:1]};<br>bins bins_interval_2 = {[7:6]};<br>bins bins_interval_3 = {[8:10]}; |
| range_interval | ^range_interval(min, max, step_size) | ^range_interval(1,a,4); | bins bins_range_interval_1 = {[‘h5:’h1]};<br>bins bins_range_interval_4 = {[‘ha:’h6]}; |
| expand | ^expand(<expr>) | ^expand({1=>2,3}, {4=>5}); | bins transition_1_2 = (1=>2);<br>bins_transition_1_3 = (1=>3);<br>bins transition_4_5 = (4=>5); |
| list_transition | ^list_transition(<expr>) | ^list_transition([2]{1=>2,3}, *{4=>5=>6}, -{7=>8} | bins bin_list0[2] = {1=>2,3};<br>illegal_bins illegal_list0 = {4=>5=>6};<br>ignore_bins ignore_list0 = {7=>8} |

| Macro | Format | Example cfg syntax | Expanded cfg syntax |
|---|---|---|---|
| one_cp_each | ^one_cp_each(<signal>) | ^one_cp_each(test)<br>test is:<br>case-1: 2 bit vector<br>case-2: enum with pass/fail | Case-1:<br>- cp test_bit_0<br>- cp_test_bit_1<br>Case-2:<br>- cp test_enum_pass<br>- cp test_enum_fail |
| one_hot | ^one_hot(<signal>) | ^one_hot(test)<br>test is a 2 bit vector | cp test_one_hot: {test}<br>bin_0(test[0]), bin_1(test[1]) ; |
| cross_expand | ^cross_expand(“cp_1,cp_2”, “cp_3, cp_4”) | ^cross_expand(“cp_1,cp_2”, “cp_3”) | cross cp_1Xcp_ 3: {cp_1,cp_3};<br>cross cp_2Xcp_3: {cp_2, cp_3}; |

# The Guiding Commandments

- Thou shalt keep **verif engineers happy.**
- Thou shalt **automate the mundane**.
- Thou shalt be **TRANSACTION-WISE**.
- Thou shalt **separate concerns**.
- Thou shalt **not repeat** thyself.
- Thou shalt be **concise**.
- Thou shalt be **flexible**.

# Results

- Robust and reusable structure
  - Transaction based TLM coverage collectors
- Versatile flow
- Consistent implementation
  - Lesser ambiguity/disruptions
- Easy reviewability of implementation.
- Reduced coding efforts
  - Code to be written reduced by 75-80%
- Happy Engineers! ☺

# Future

- Improvisations: patterns, parameterisations
- Leverage Verific
  - For transaction details

# Questions?