2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Adaptive Test Generation for Fast Functional Coverage Closure

Azade Nazi, Qijing Huang, Hamid Shojaei, Hodjat Asghari Esfeden, Azalia Mirhoseini, Richard Ho

Google Research

accellera SYSTEMS INITIATIVE

# Outline

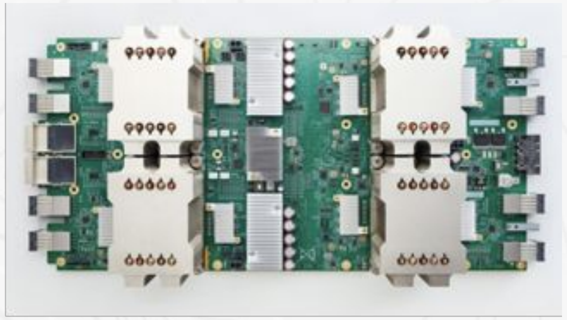
- Main Goal in Google
  - Improve chip design flow
- Challenges in standard Constraint Random Verification flow
- Introduce Smart Constraint Solver
  - Propose CDG4CDG
  - Scalability
- Experimental Results
  - Achieved up to 21.7× speedup in a fully automated flow.
- Conclusion

# Using AI/ML to Accelerate Design Flows

**Goal**

Develop scalable, and generalizable machine learning framework with rapid evaluation and turn-around time to shorten the chip design process.

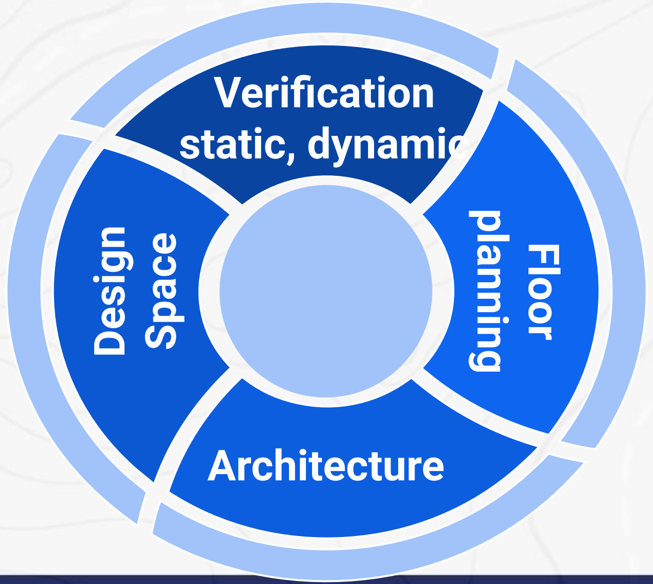
Cloud ML Engines

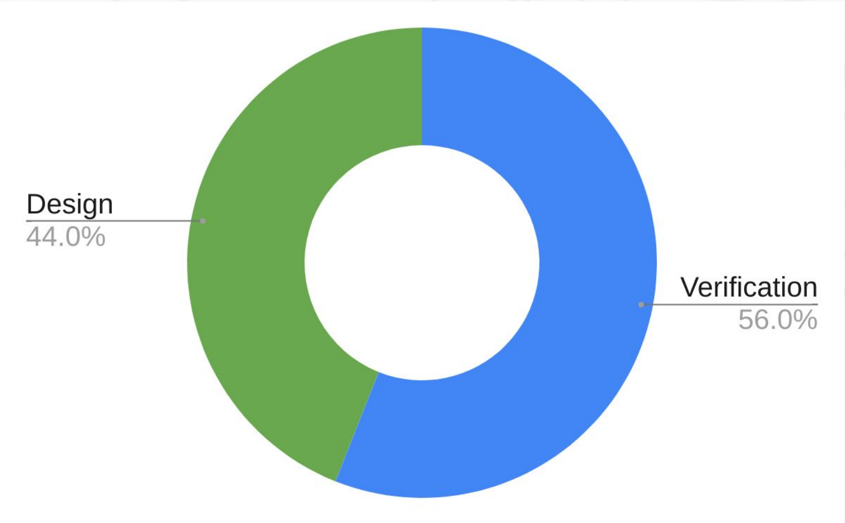


Design More Efficient Accelerator

**Research Direction**

- Apply ML into four different stages of chip design

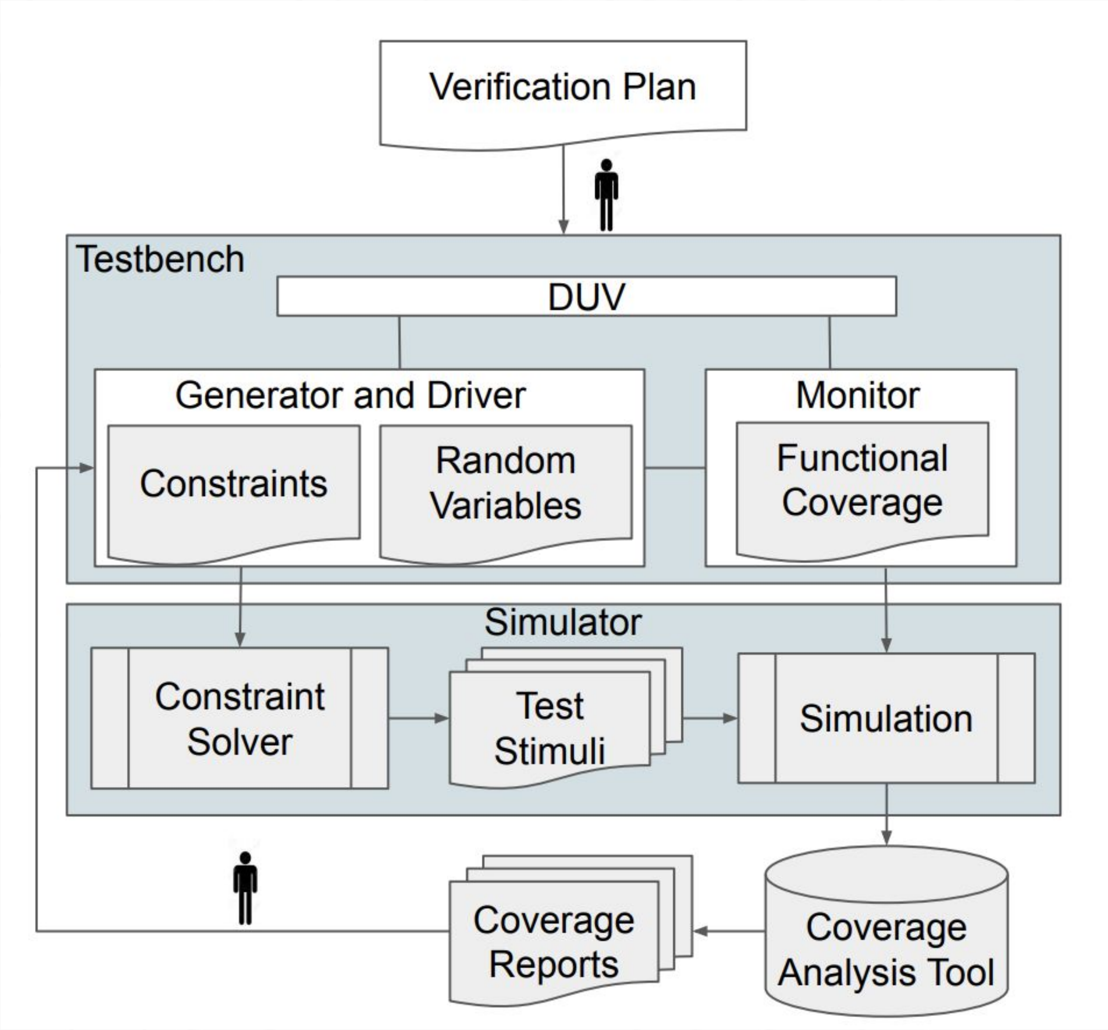



Source: Wilson Research Group/Mentor

# Standard Constraint Random Verification Flow

- Verification of complex designs starts with the definition of a verification plan
- Verification engineers create a testbench to simulate the designs at the code level
    - Checker verifies the design output against the modeled output
- Test Stimuli is generated from the feasible solutions of the constraint solver



# Constraints and Coverage

- Distribution Constraint

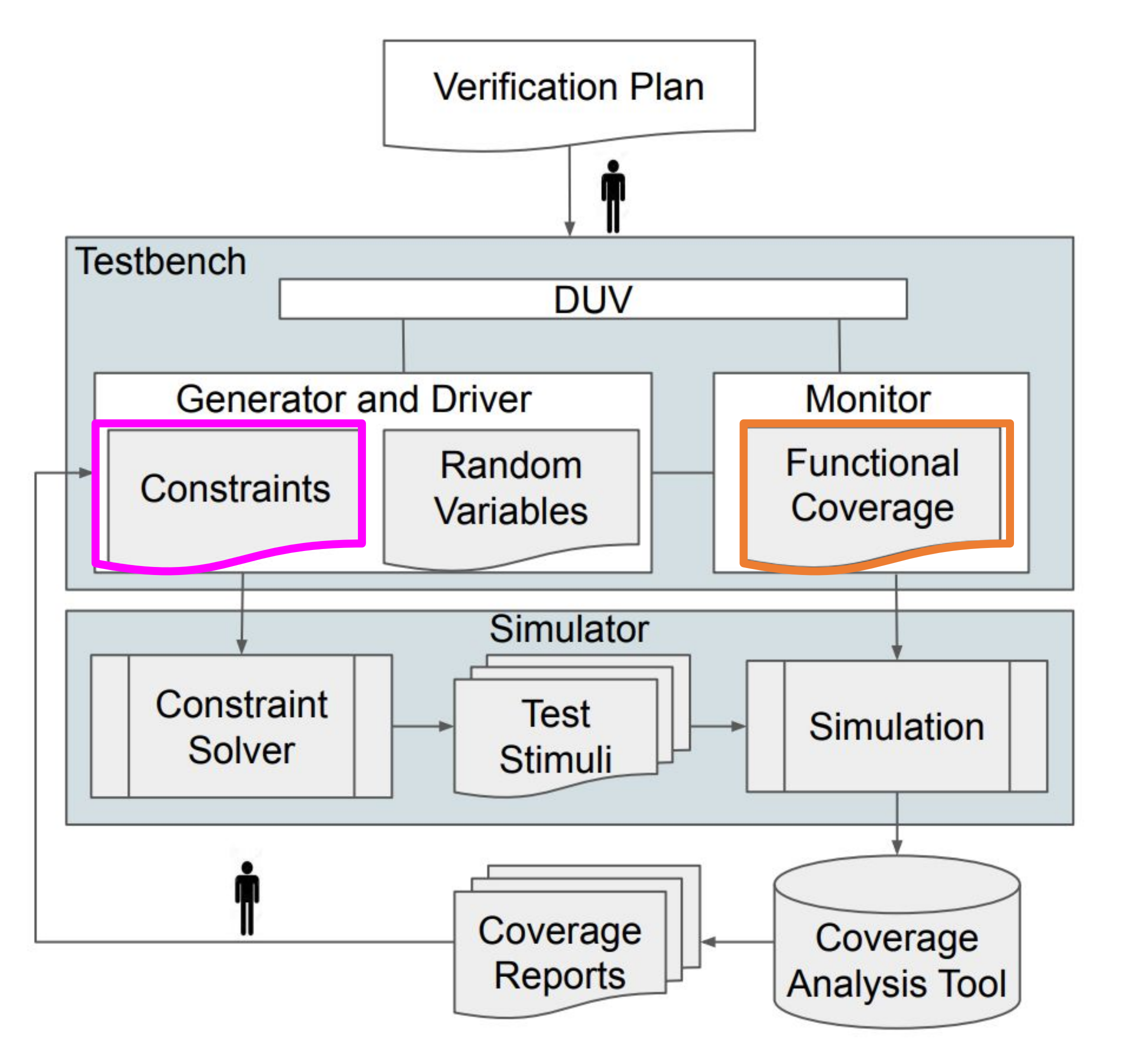
```
rand bit [1:0] State;
constraint con_State {
    State dist {
        idle := 10,
        start := 10,
        read := 10,
        write := 10
    };}
```

- Legalization Constraint

```
constraint c {
  // inclusive
  src_port inside { [8'h0:8'hA],8'h14,8'h18 };
  // exclusive
  ! (des_port inside { [8'h4:8'hFF] });
}
```

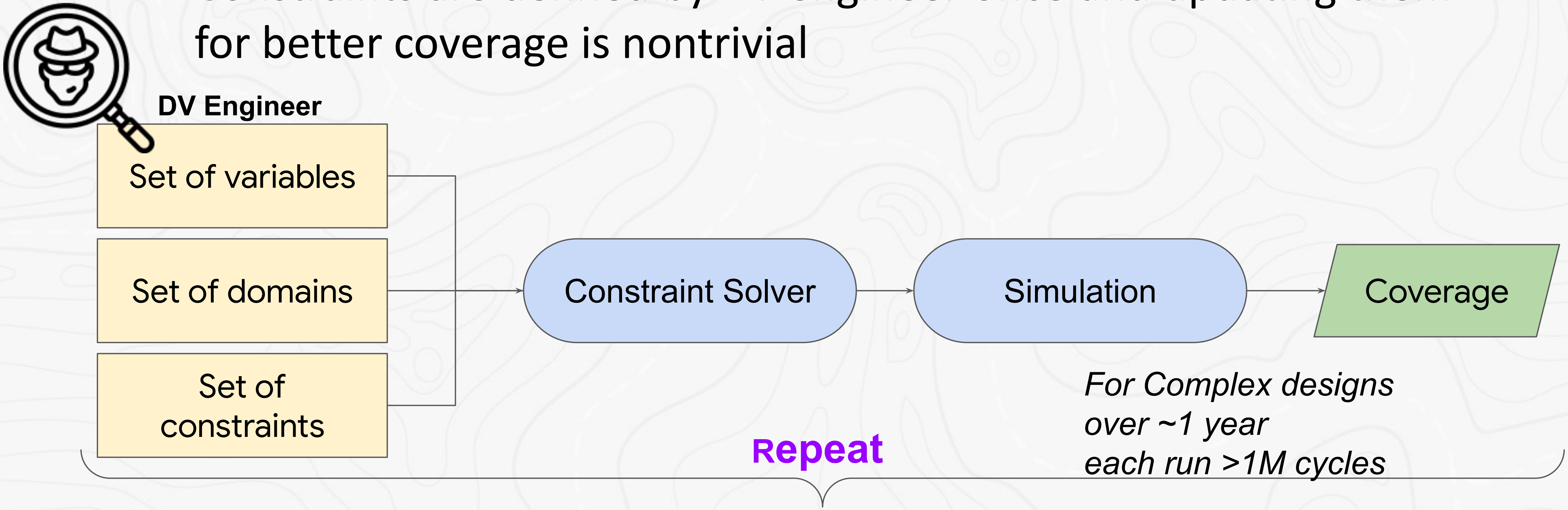
- Ordering Constraint

```
constraint frame_sizes {
  solve zero before data.size;
  zero -> data.size == 0;
  data.size inside {[0:10]};}
```

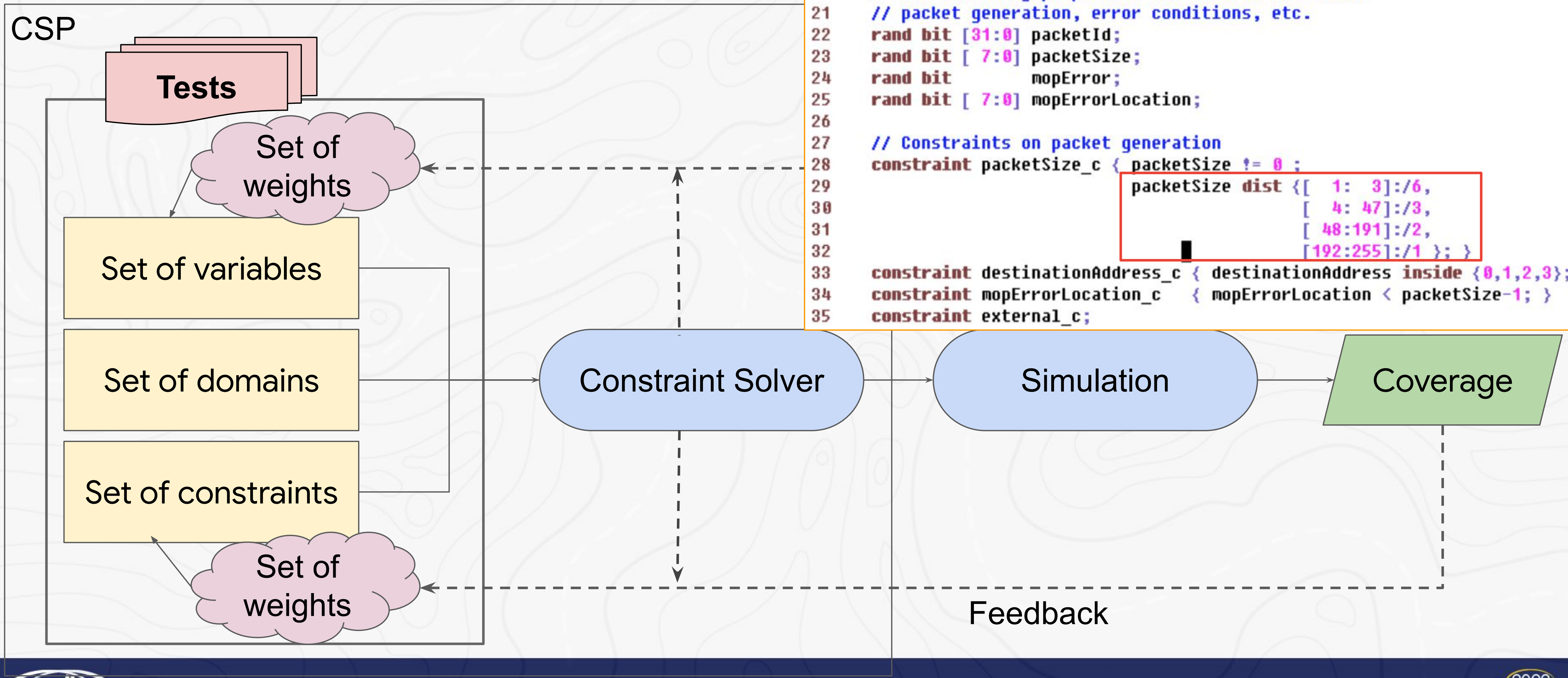


# Design Sign-off: Coverage Closure

- CRV mostly relies on randomness
  - At every simulation cycle the seed is changed
- Constraints are defined by DV engineer once and updating them for better coverage is nontrivial



# Smart Constraint Solver



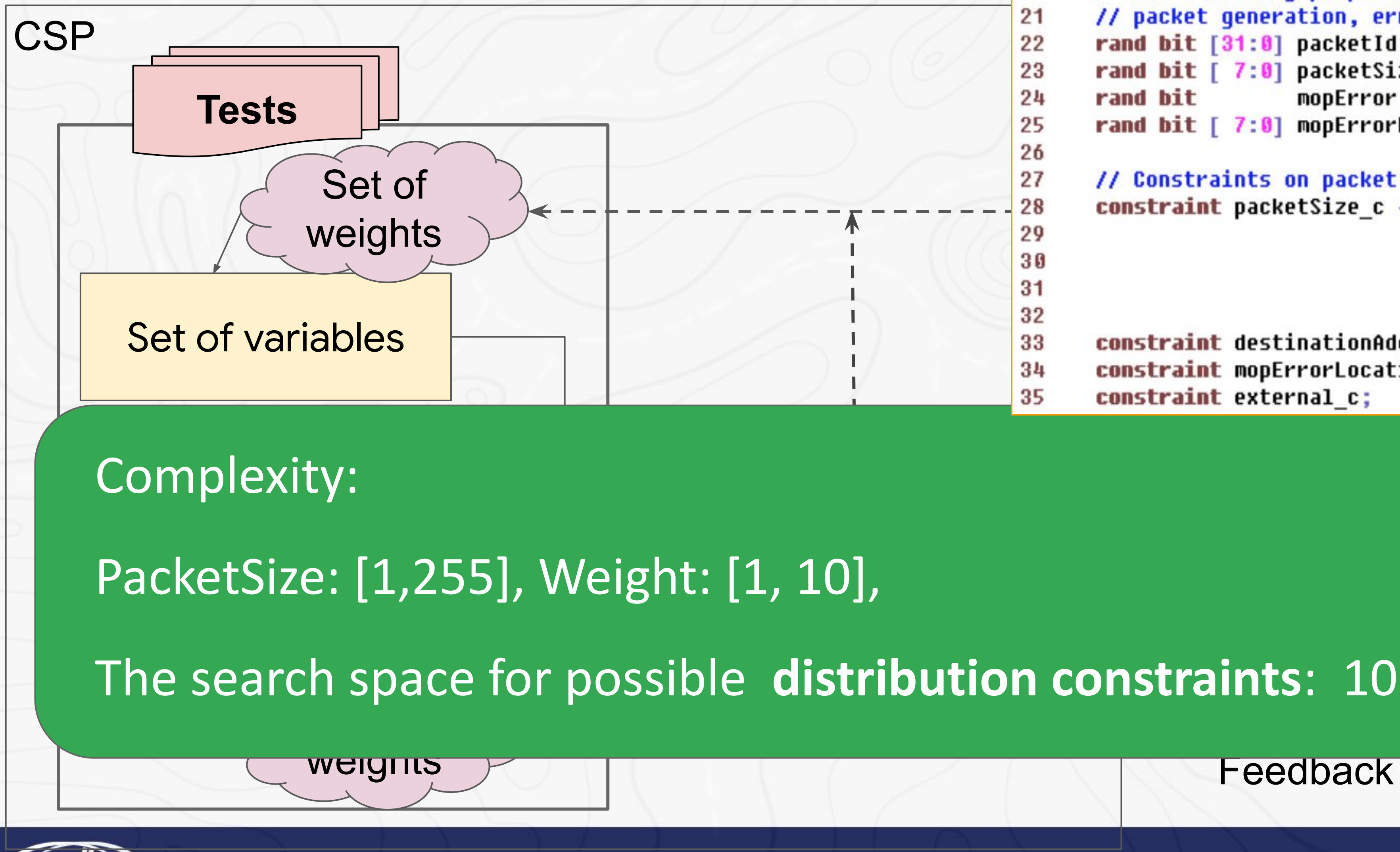

```
class packet;

    // The following properties are visible to the DUT
    rand bit [7:0]  destinationAddress;
    rand bit [7:0]  sourceAddress;
         bit [7:0]  packetData[$];

    // The following properties are used to influence
    // packet generation, error conditions, etc.
    rand bit [31:0] packetId;
    rand bit [ 7:0] packetSize;
    rand bit        mopError;
    rand bit [ 7:0] mopErrorLocation;

    // Constraints on packet generation
    constraint packetSize_c { packetSize != 0 ;
                              packetSize dist {[   1:   3]:/6,
                                               [   4:  47]:/3,
                                               [  48:191]:/2,
                                               [192:255]:/1 }; }
    constraint destinationAddress_c { destinationAddress inside {0,1,2,3};}
    constraint mopErrorLocation_c   { mopErrorLocation < packetSize-1; }
    constraint external_c;
```

# Smart Constraint Solver




```
class packet;

   // The following properties are visible to the DUT
   rand bit [7:0]  destinationAddress;
   rand bit [7:0]  sourceAddress;
        bit [7:0]  packetData[$];

   // The following properties are used to influence
   // packet generation, error conditions, etc.
   rand bit [31:0] packetId;
   rand bit [ 7:0] packetSize;
   rand bit        mopError;
   rand bit [ 7:0] mopErrorLocation;

   // Constraints on packet generation
   constraint packetSize_c { packetSize != 0 ;
                             packetSize dist {[  1:  3]:/6,
                                              [  4: 47]:/3,
                                              [ 48:191]:/2,
                                              [192:255]:/1 }; }
   constraint destinationAddress_c { destinationAddress inside {0,1,2,3};}
   constraint mopErrorLocation_c   { mopErrorLocation < packetSize-1; }
   constraint external_c;
```

Complexity:

PacketSize: [1,255], Weight: [1, 10],

The search space for possible **distribution constraints**: 10^252

# Related Work: Coverage-directed Test Generation

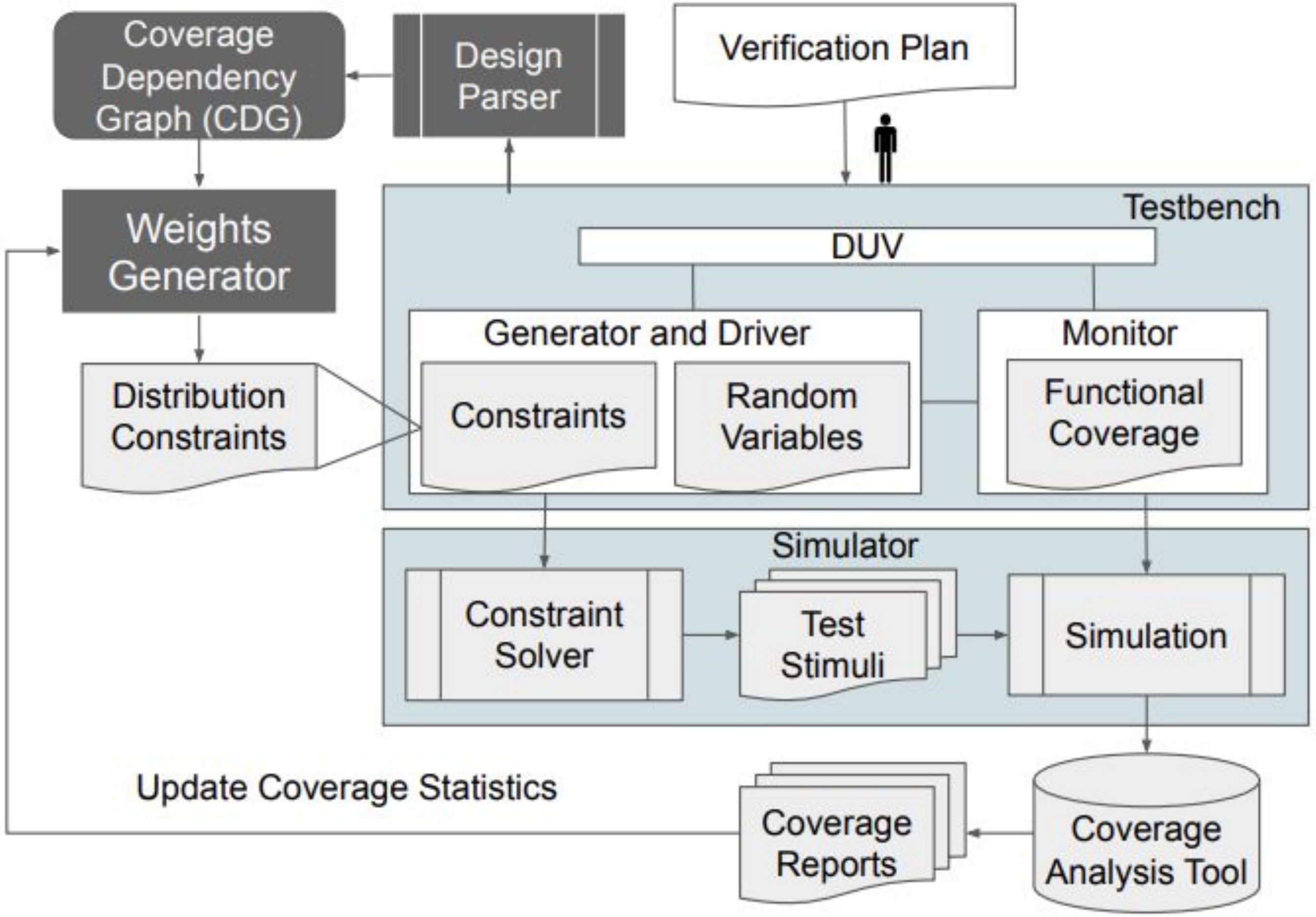
- Model-driven
  - Converting the design into an abstract finite-state machine
  - Mainly suffers from the need to create and maintain an accurate model
- Data-driven
  - Models the relationships between coverage and stimuli directives from the simulation feedback
    - Use observations to build and train a model that captures the casualty between input variables and coverage

Limitations:

- Rely on a significant amount of domain knowledge
- Require a considerable number of simulations to obtain enough training data
- Very challenging for modern large-scale designs as data collection and model training may take several months.

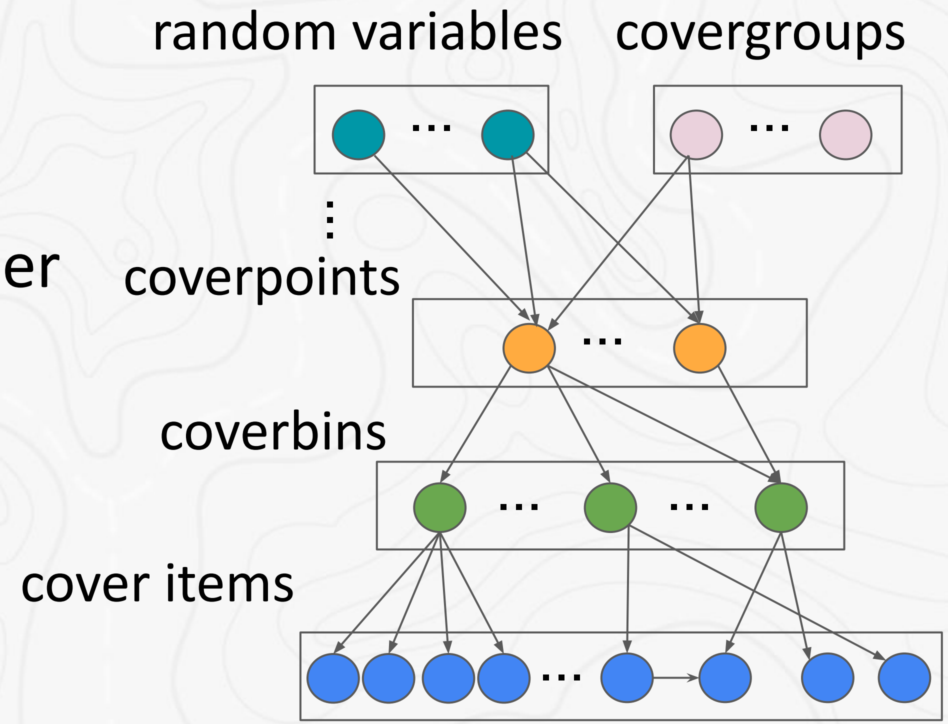
# CDG4CDG: Coverage Dependency Graph for Coverage Driven Test Generation

- Coverage Dependency Graph (CDG) Extraction
  - Build Bayesian Network automatically
- Find conditional probabilities
- Generate/update constraints by statistical inference on CDG

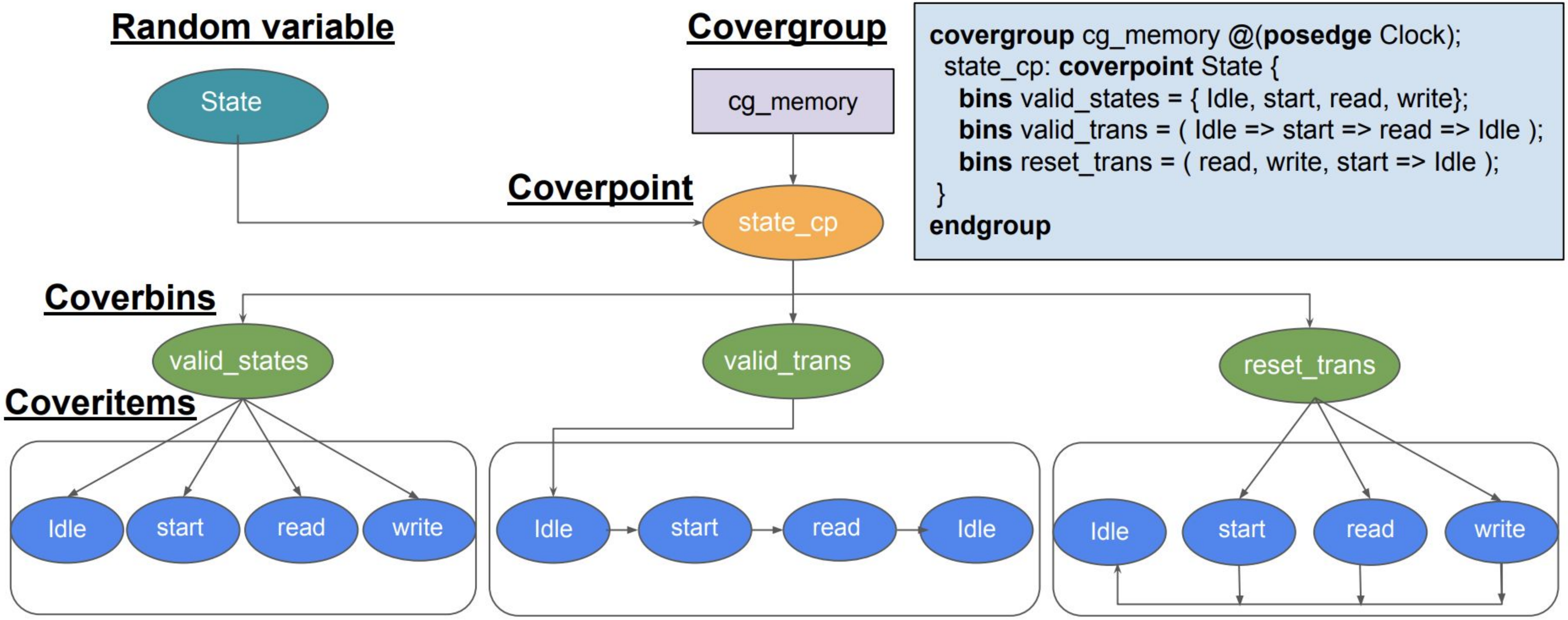


# CDG: Model Functional Coverage as a Bayesian Network

- We use design parser to find the correlation of the cover items and the random variables
  - For each coverpoint we query the parser until we reach a random variable
  - For each coverbin, the coveritems are the values of the random variables required to be sampled in a test



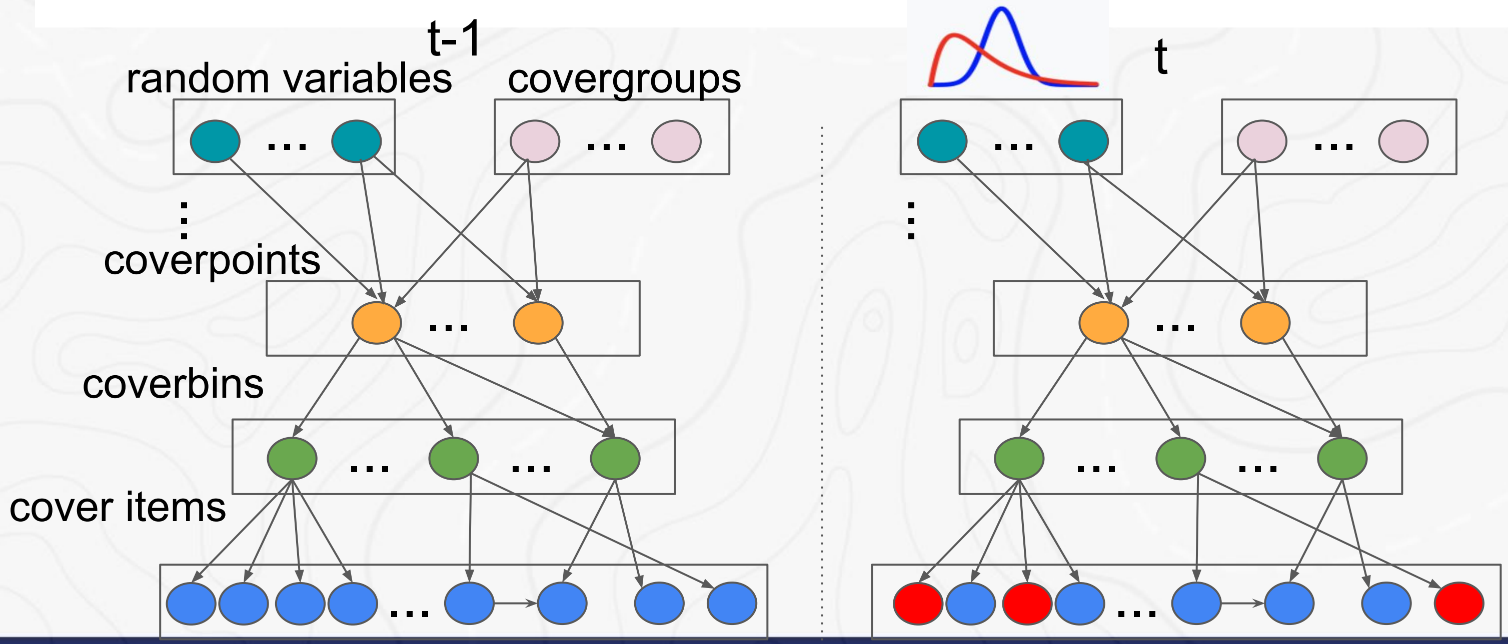
# CDG Example




```
covergroup cg_memory @(posedge Clock);
 state_cp: coverpoint State {
  bins valid_states = { Idle, start, read, write};
  bins valid_trans = ( Idle => start => read => Idle );
  bins reset_trans = ( read, write, start => Idle );
 }
endgroup
```

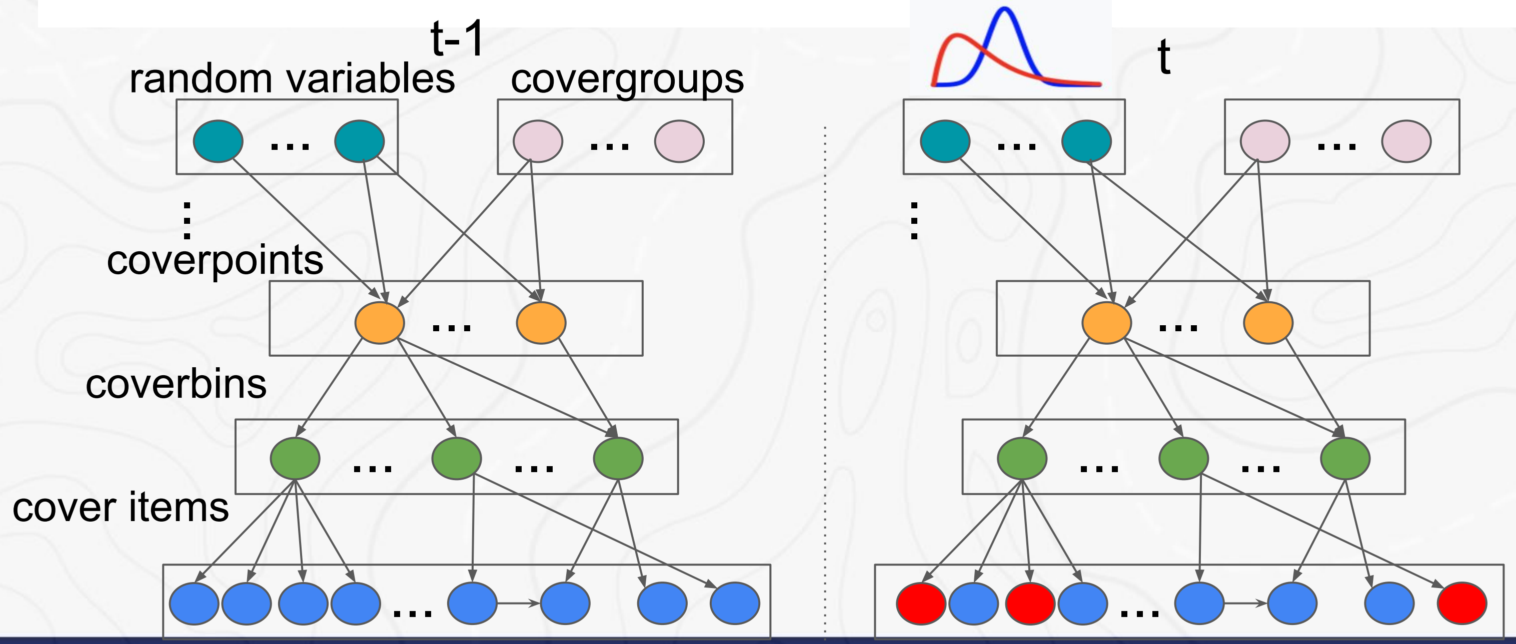
# end-to-end algorithm

1: Query testbench modules and automatically extract the CDG graph
2: **while** ! Coverage Closure **do**
3: Run simulation and get the coverage feedback
4: Estimate conditional probabilities for nodes in CDG graph
5: Update CDG graph by pruning cover items that are covered
6: for coverage holes $x$ update distribution constraints by solving $\theta_{MLE} = argmax_{\theta} P(x;\theta)$
7: **end while**



# end-to-end algorithm

1: Query testbench modules and automatically extract the CDG graph
2: **while** ! Coverage Closure **do**
3:     Run simulation and get the coverage feedback
4:     Estimate conditional probabilities for nodes in CDG graph
5:     Update CDG graph by pruning cover items that are covered
6:     for coverage holes $x$ update distribution constraints by solving $\theta_{MLE} = argmax_{\theta} P(x;\theta)$
7: **end while**

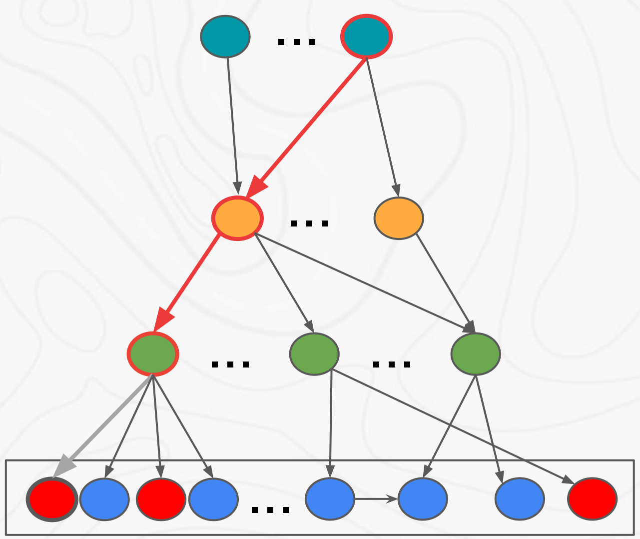



Scalability?

# Our Approach to Handle Scalability Challenges

Challenge 1: Exact inference in Bayes nets is NP-hard [1]

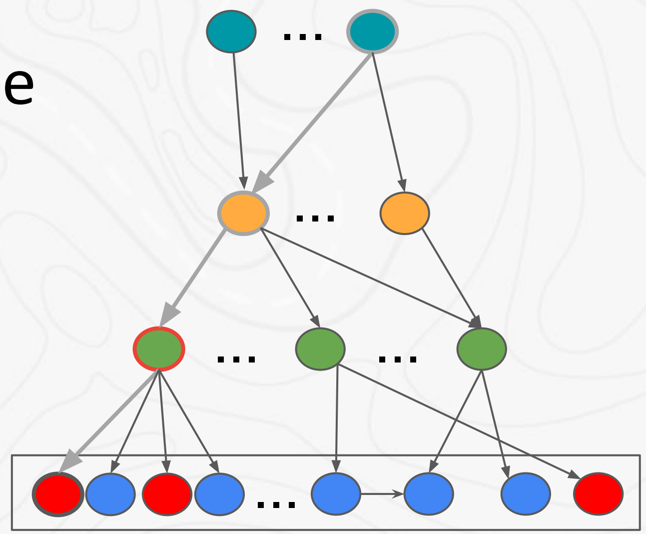
- Our CDG structure is polytree, thus belief propagation performs inference efficiently linear in number of nodes [2]
  - We use simulation feedback to calculate probability of hit/no hit for each coverpoint
  - We leverage the polytree property since there is only one path from a coverpoint to the variables
  - We iteratively update distribution constraints by aggregation of no hit counts




[1] Cooper, G. F., 1990. The computational complexity of probabilistic inference using Bayesian belief networks.
[2] Bayesian Artificial Intelligence (2004), Kevin B. Korb and Ann E. Nicholson, Chapman and Hall, CRC Press

# Our Approach to Handle Scalability Challenges

Challenge 2: Modern large-scale designs may have over thousands of variables and adding constraints over each variable significantly hampers practicality of the constraint solver




Solution

- We pick top-k variables from CDG graph
  - Rank variables by the number of coverpoints related to them from its induced subgraph
  - Define distribution over range of values

# Evaluation Setup

Baselines

- default_dist
  - Existing distribution (if any)
- random_dist
  - Change the distribution constraints randomly

Designs

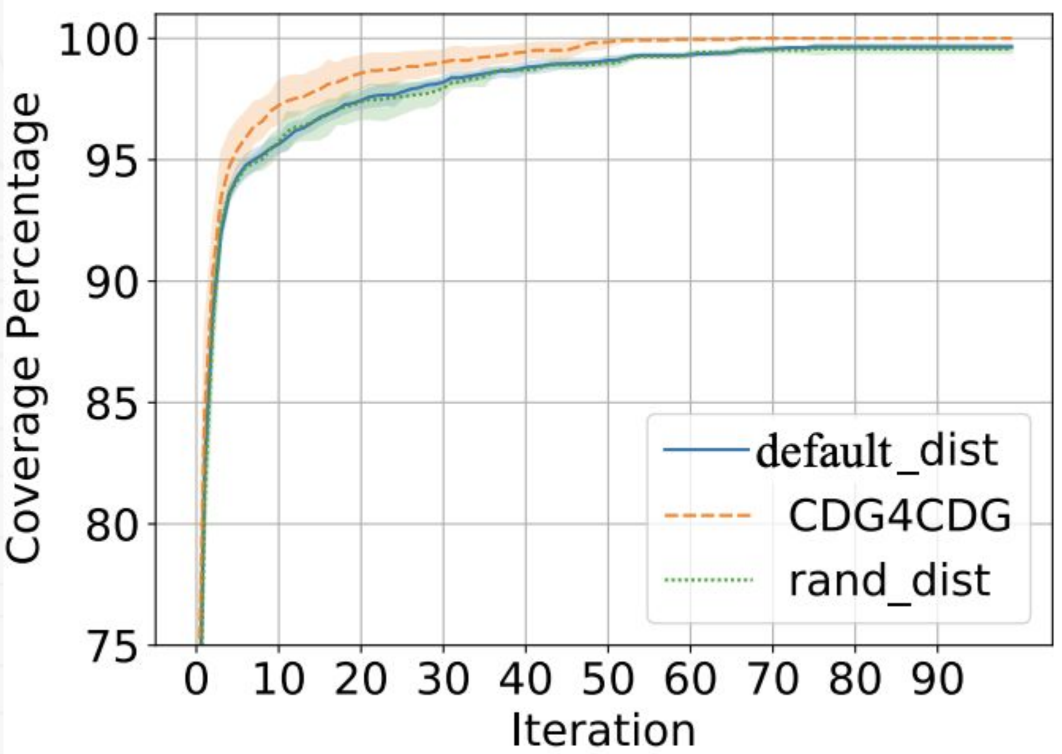
- Industrial Accelerator SML: TPU block
  - TPU block consists of multiple large and complex designs
  - Considered three designs in this TPU block (SML1-3)
- RISCV-Ibex

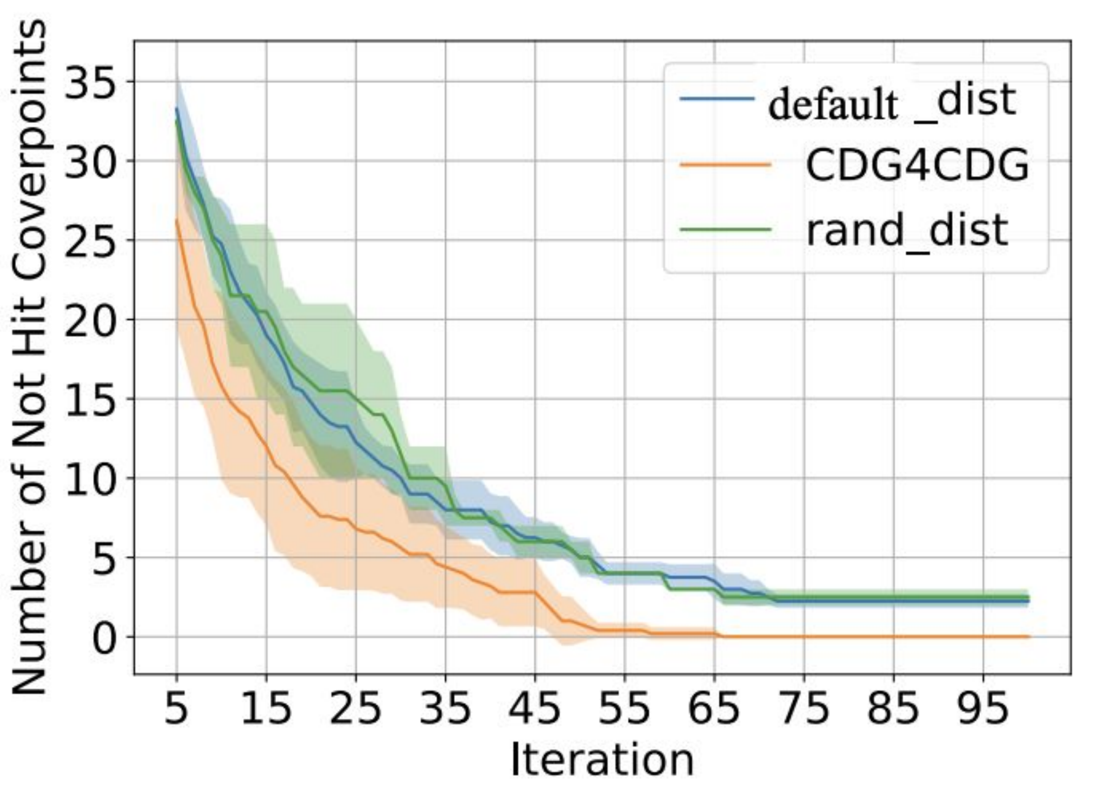
Experimental Results:

- Evaluate CDG4CDG
- Theoretical bound on baseline + Empirical result
- Impact on code coverage

# TPU: SML 1

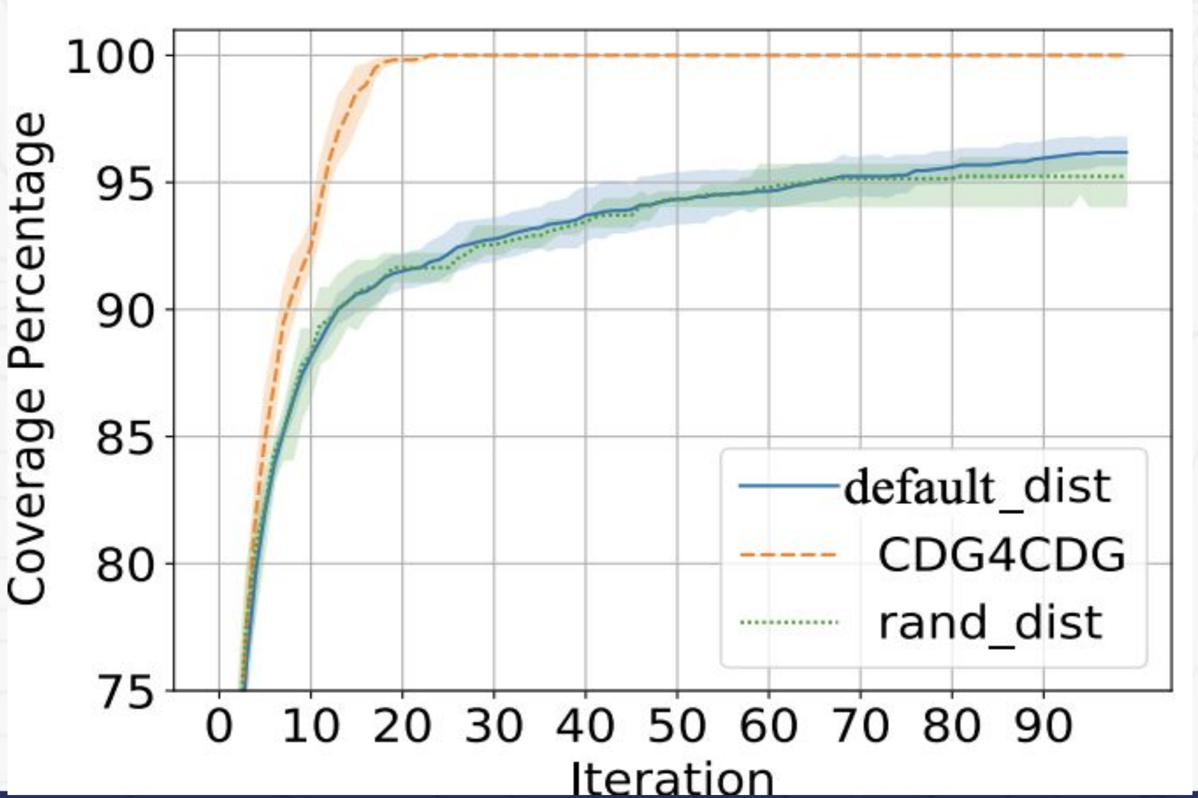
| | #coverpoints | #random variables | Coverage closure | #bundles | Speed up |
|---|---|---|---|---|---|
| SML1 | 371 | 5 ~ 10^(2^22) | 66 vs 340 | 528 vs ~3k | =5.15x |

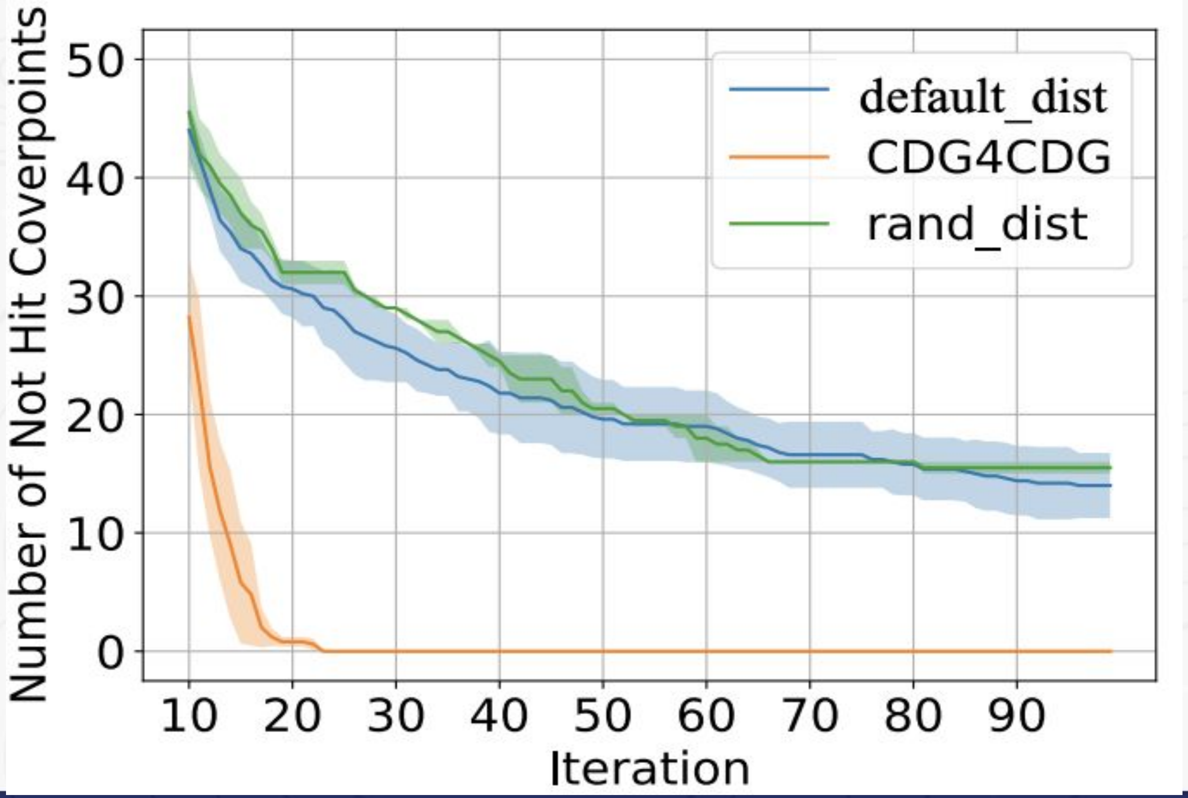


# TPU:  SML 2

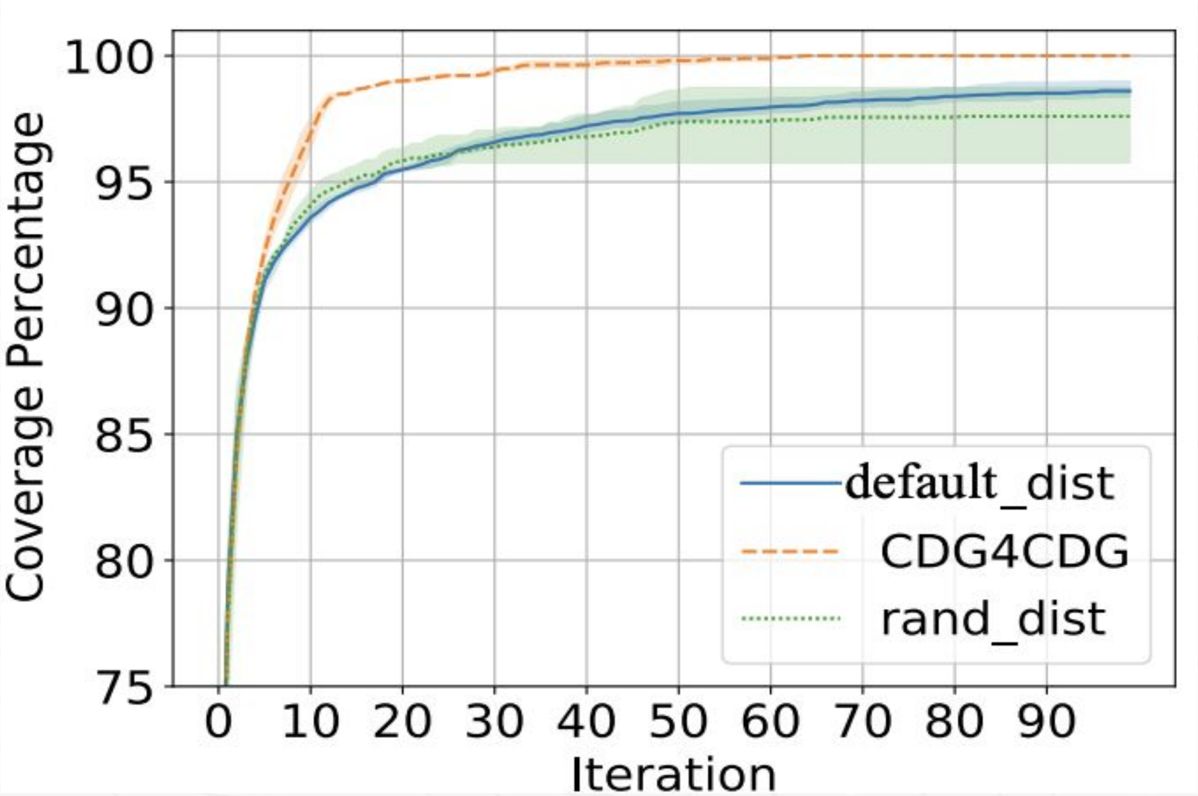
| | #coverpoints | #random variables | Coverage closure | #bundles | Speed up |
|---|---|---|---|---|---|
| SML 2 | 574 | 7, ~10^(2^43) | 23 vs 500+ | 184 vs 4k | >21.7x |

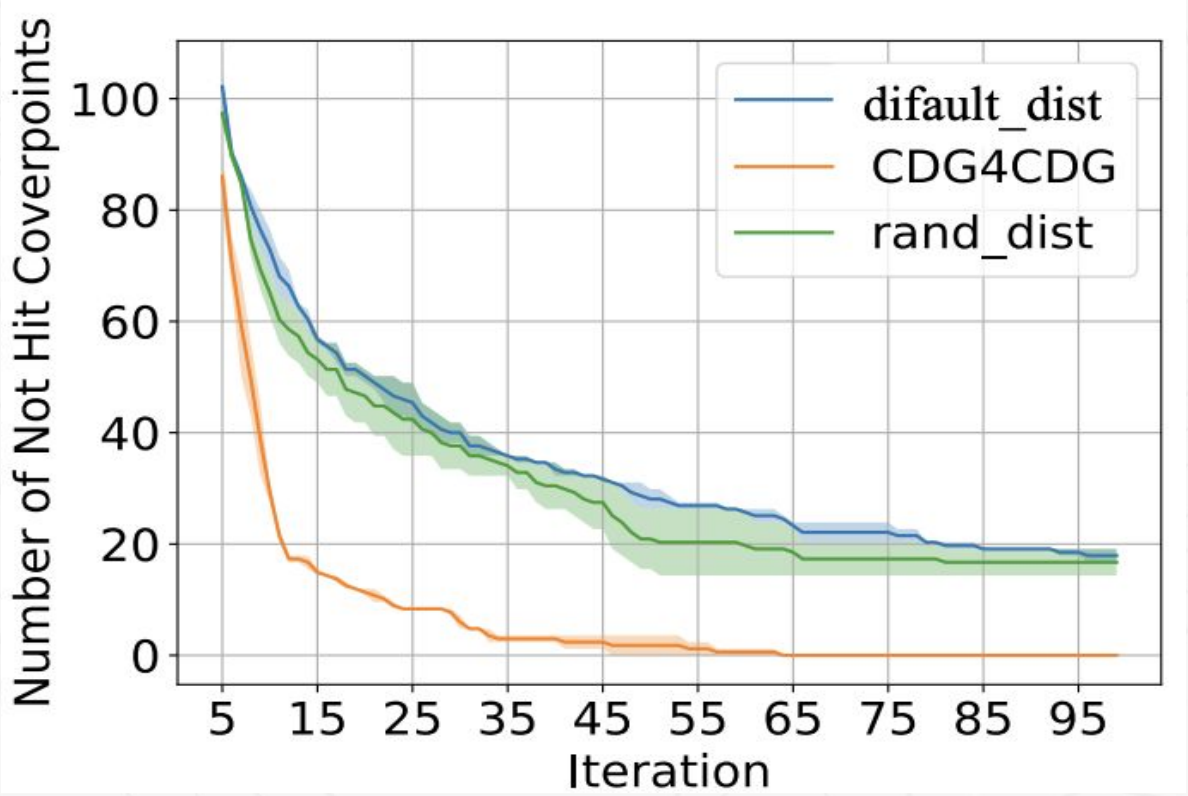


# TPU: SML 3

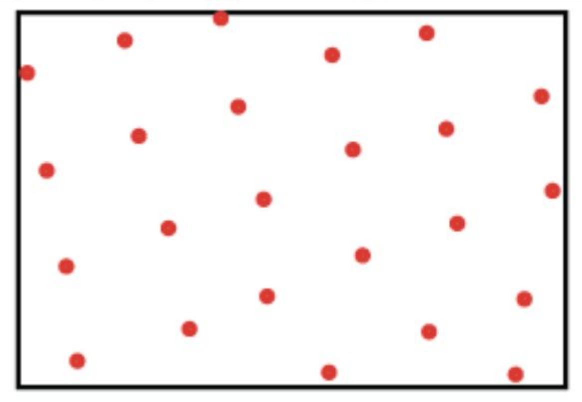
| | #coverpoints | #random variables | Coverage closure | #bundles | Speed up |
|---|---|---|---|---|---|
| SML 3 | 1129 | 12, ~10^(2^65) | 66 vs 500+ | 504 vs 4k | >7.9x |

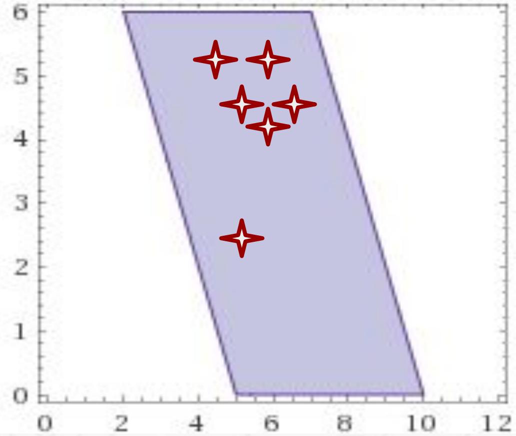


# Bound on default_dist Baseline

For the baseline, the lower bound of the expected number of simulation runs to cover all m coveritems is O(m log m), while the upper bound is infinite

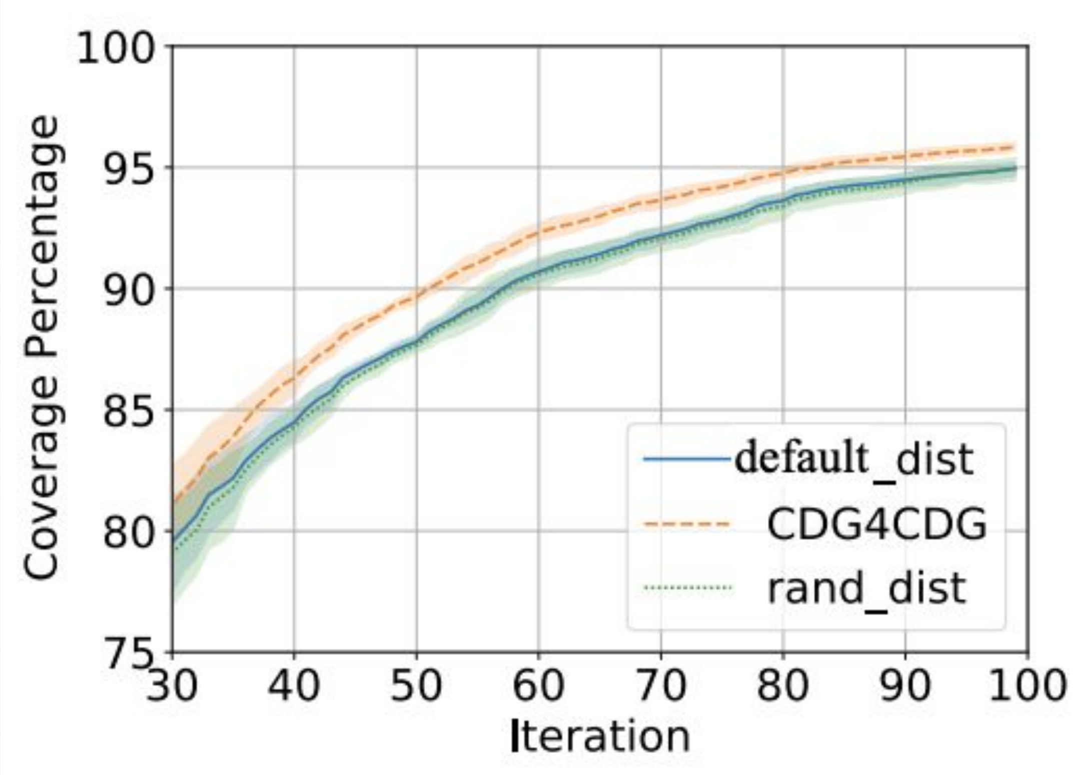


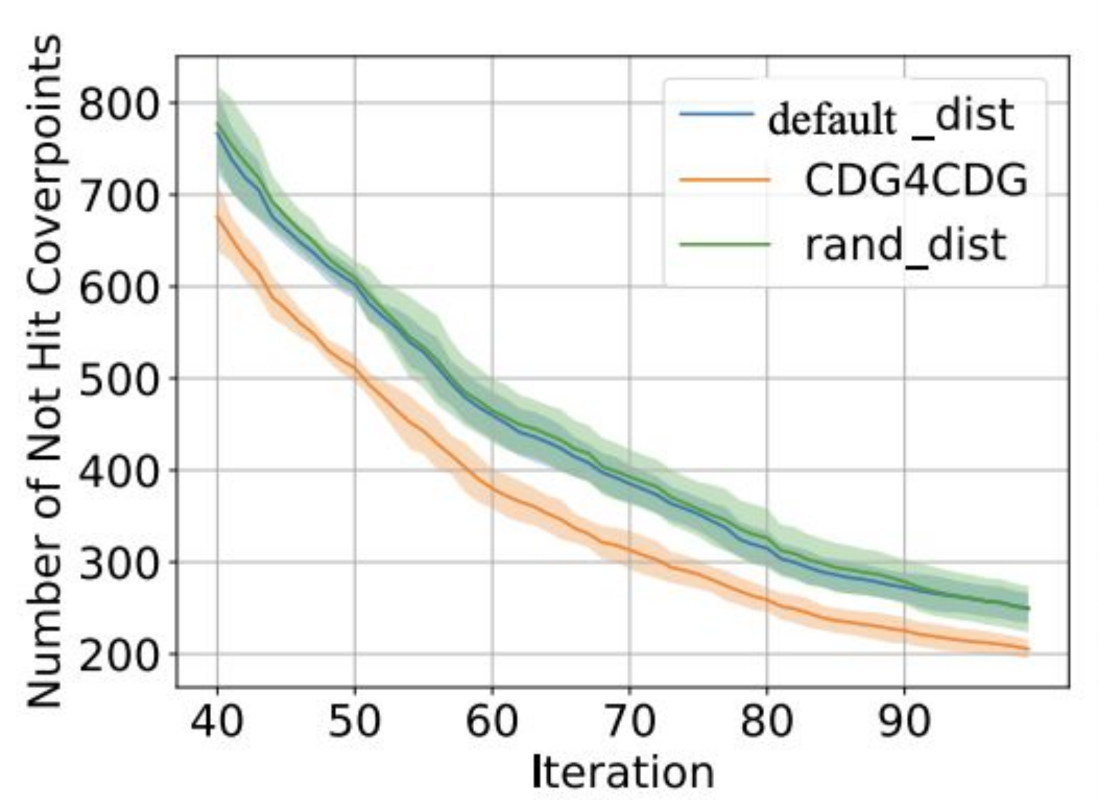



- The best scenario is when the distribution constraint is uniform and coveritems are equally likely to be covered. In this case, the problem is reduced to an instance of well known Coupon collector’s problem

# Empirical results

Most of the coveritems in RISCV-Ibex are uniformly specified among all random variable values

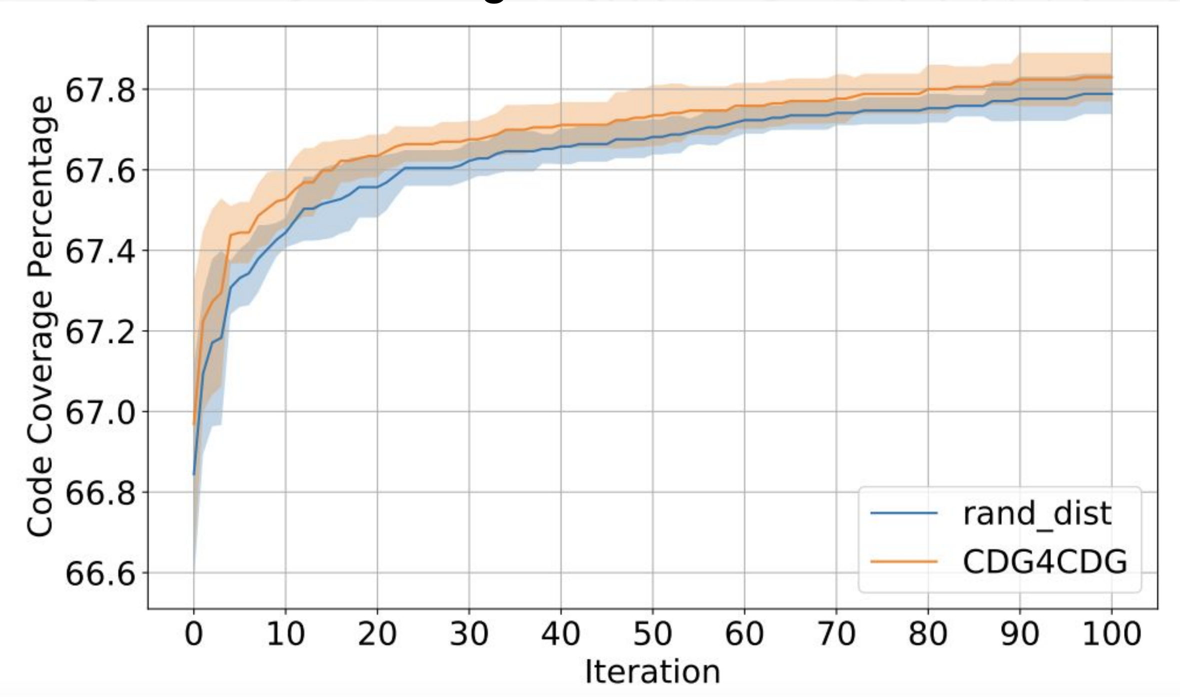


# Impact of Functional Coverage on Code Coverage

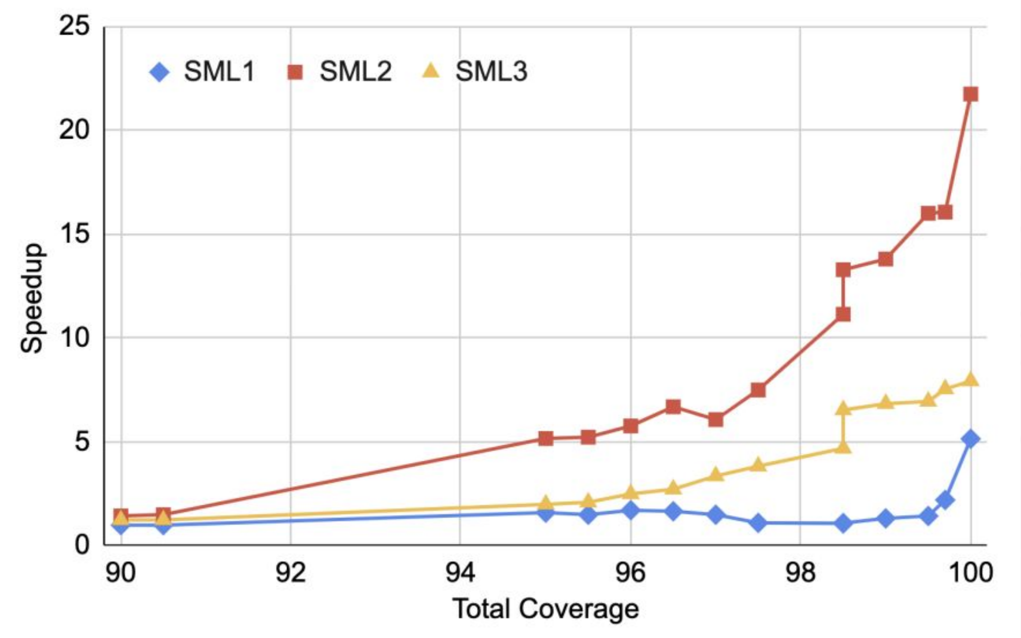
Though CDG4CDG is designed to improve the functional coverage, it alters the variable values and the execution paths of the simulation, and subsequently may affect code coverage as well.

Code coverage on the RISCV-Ibex



# Conclusion

- We introduced an automated test generation framework, CDG4CDG, to accelerate the functional coverage convergence
- CDG4CDG achieves consistent coverage improvement
- Reaches the coverage closure significantly faster with up to 21.7× speedup in a fully automated flow



# Questions

azade@google.com
hamids@google.com