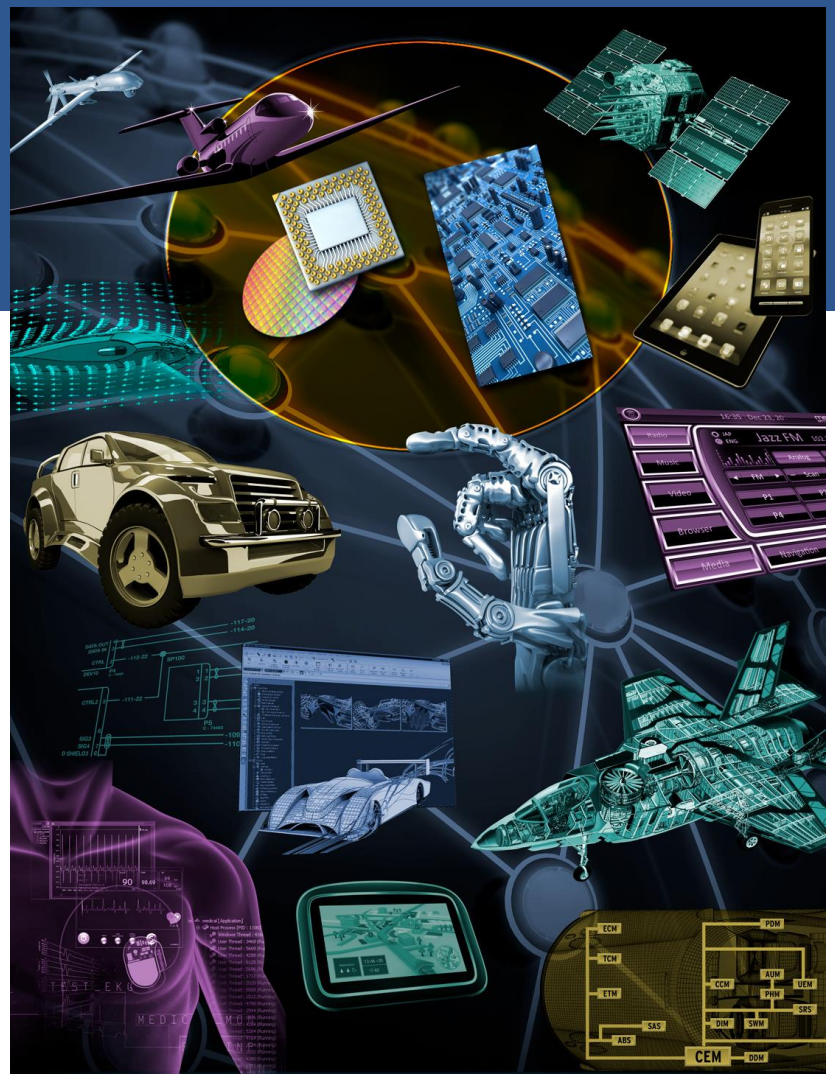


빠르고 강력하게 검증생산성 높이기 - 퀘스타 인팩트를 활용한 커버리지 클로져 달성

박 성진 대리

Application Engineer

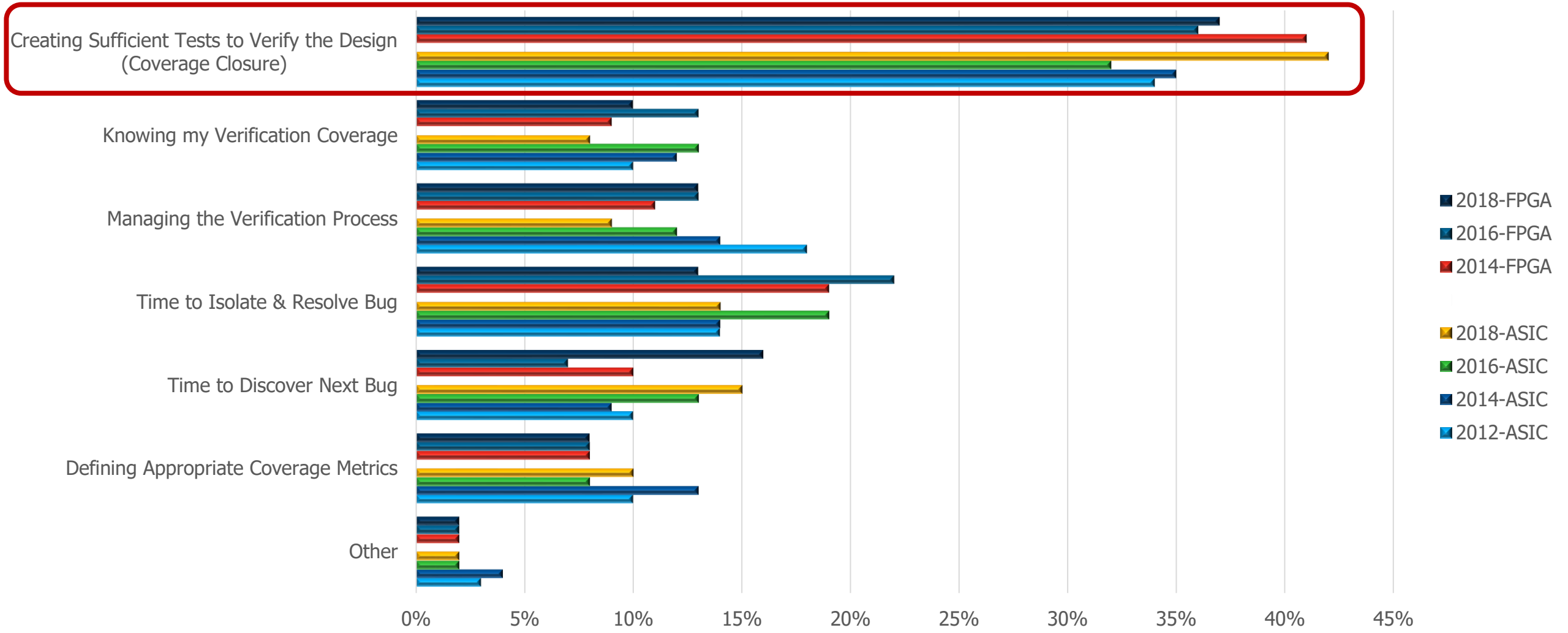
Mentor, A Siemens Business

WT PacRim Front End Solution

Mentor®
A Siemens Business

Industry Trends

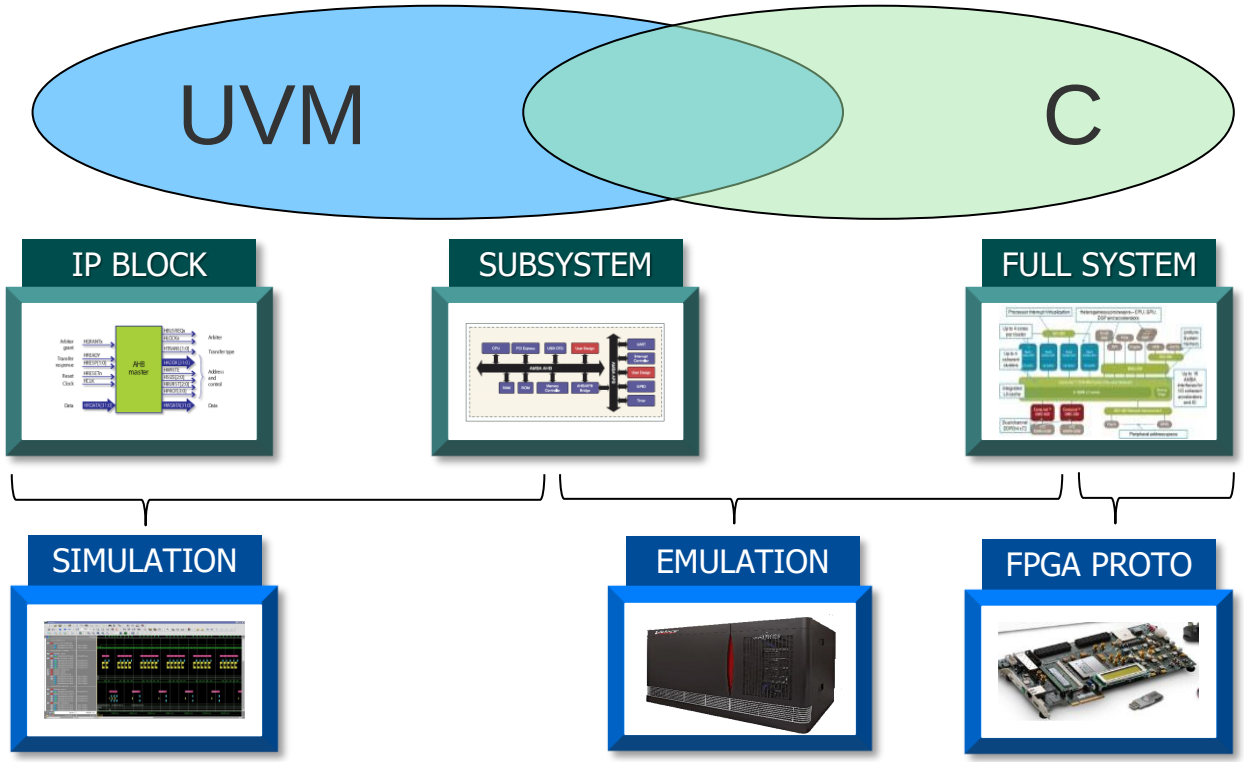
Biggest Verification Challenges



Source: Wilson Research Group and Mentor Graphics, 2016 Functional Verification Study

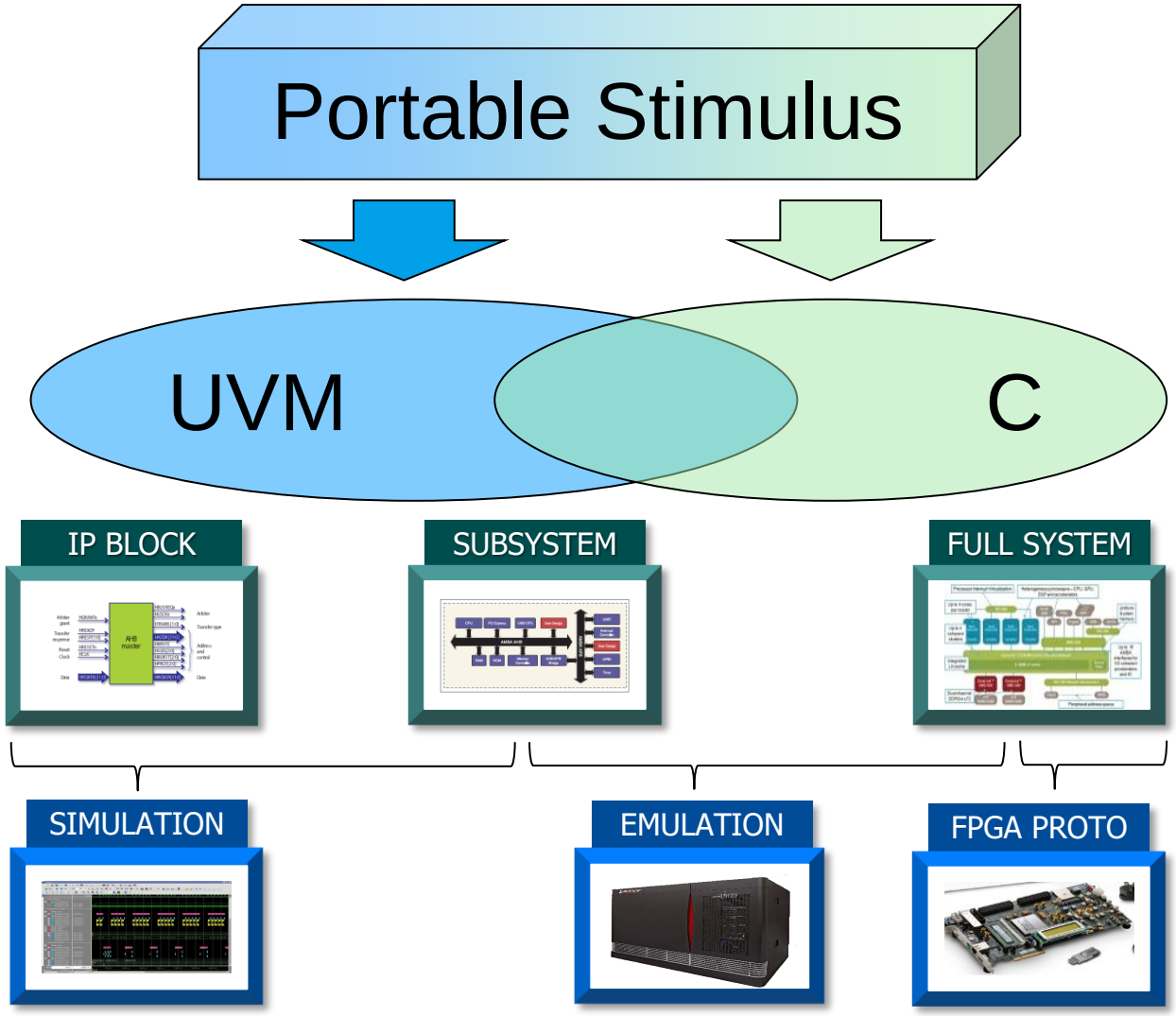
Test Creation Challenges

- Directed tests
  - Low productivity
  - Difficult to account for complex unforeseen interactions
- Constrained-random tests
  - Requires significant user guidance
  - Many scenarios exercised outside of coverage goals
  - Don't (easily) scale to scenario level
- Need support across many test environments
  - C++ for high-level synthesis
  - UVM block-level and subsystem
  - Embedded software SoC
- Must productively create high-quality tests
  - Find bugs early
  - Maximize stimulus-space coverage
  - Generate required stimulus with minimum user guidance



Portable Stimulus Vision

- Single specification of test intent
  - With capabilities supporting block to SoC
  - Support for multiple engines
- Capture *scenario space*
  - Data space
  - Operational sequence
  - Resource dependencies
- Enable automated test generation
  - Goal-driven stimulus generation
  - Focus on *what* to test, not *how* to create tests
  - Target stimulus across environments



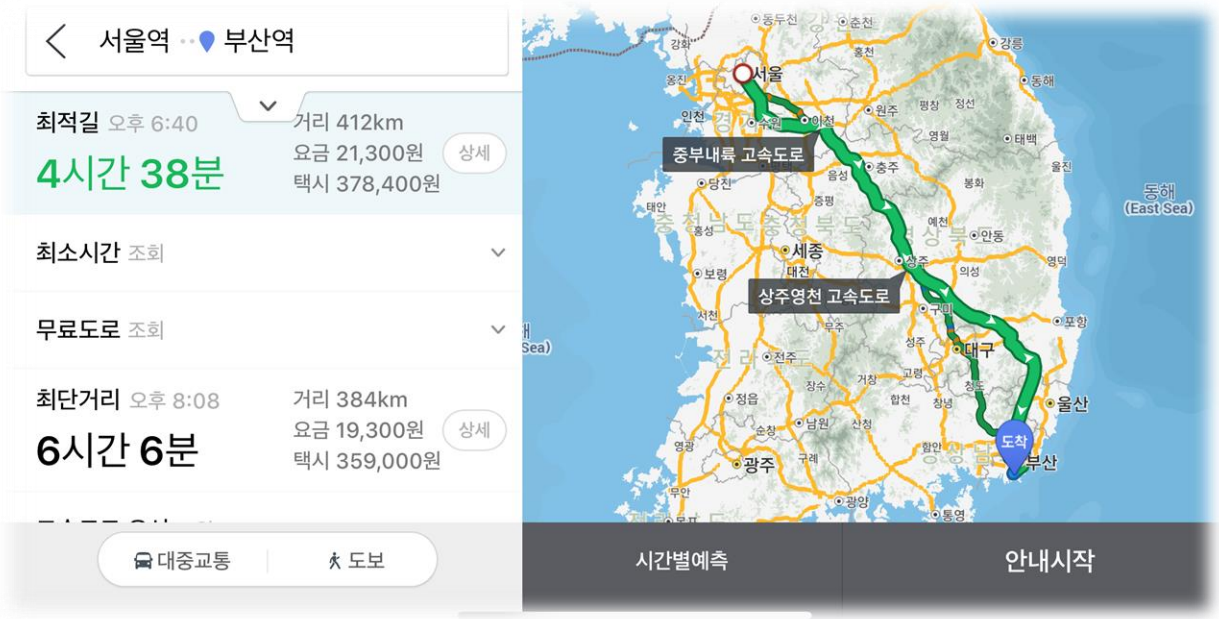
Mentor® A Siemens Business

What vs. *How*

- Most SoC tests are directed
- Manually determining turn-by-turn directions
- Hard to account for new stops
- Route limited by driver's biases

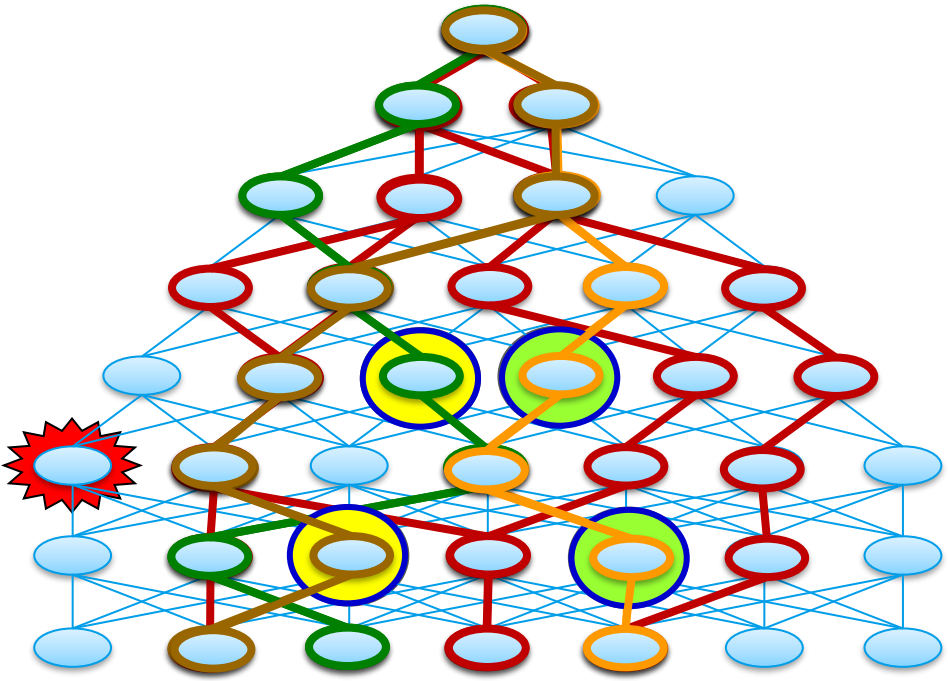
- A declarative description enables automation and analysis
- Automation makes test intent portable
- Enables retargeting to different environments
- Automation makes test techniques portable
- Bring automated constraint-driven tests to SoC

- Declarative tests let the tool do the work
- Explore all possible options
- Easy to optimize
- Guided by preferences



Constrained-Random and Functional Coverage

Passive Coverage

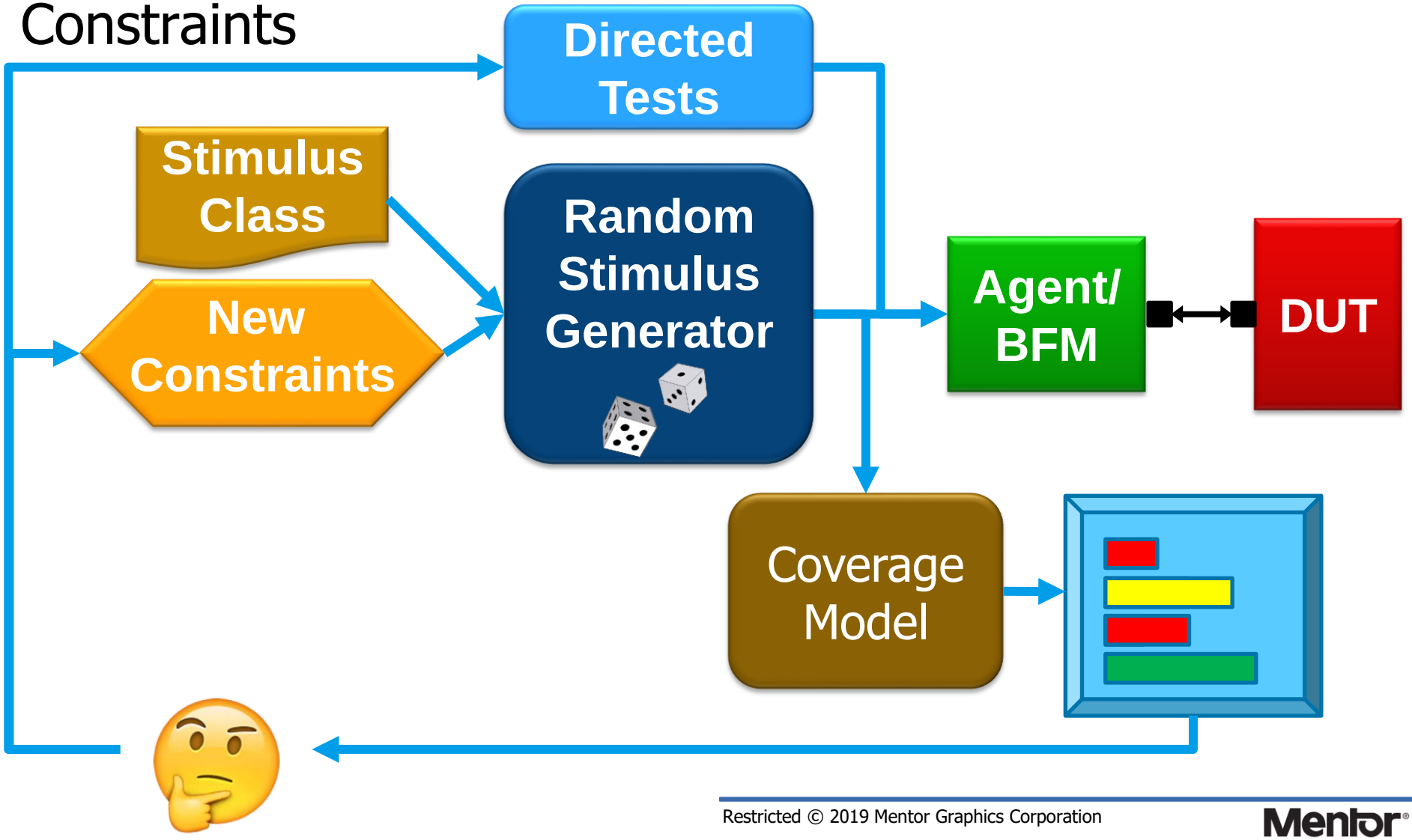


7

- Functional Coverage defines critical states
- C-R tests hits those states in unpredictable ways
- Often C-R tests repeat states or miss coverage points
- Verification Management identifies which tests actually hit the coverage points

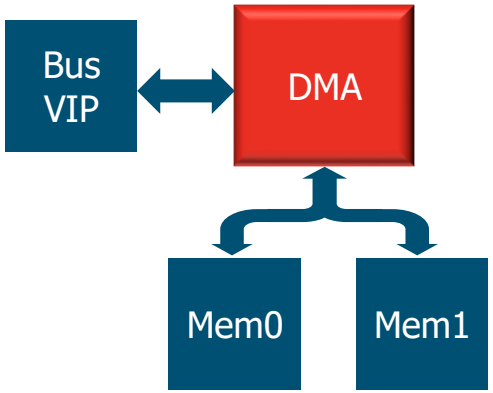
Closing Coverage with Constrained-Random

- Self-tweeking Constraints are a Myth
- Requires Manual Analysis & Adjustment



Existing UVM Environment

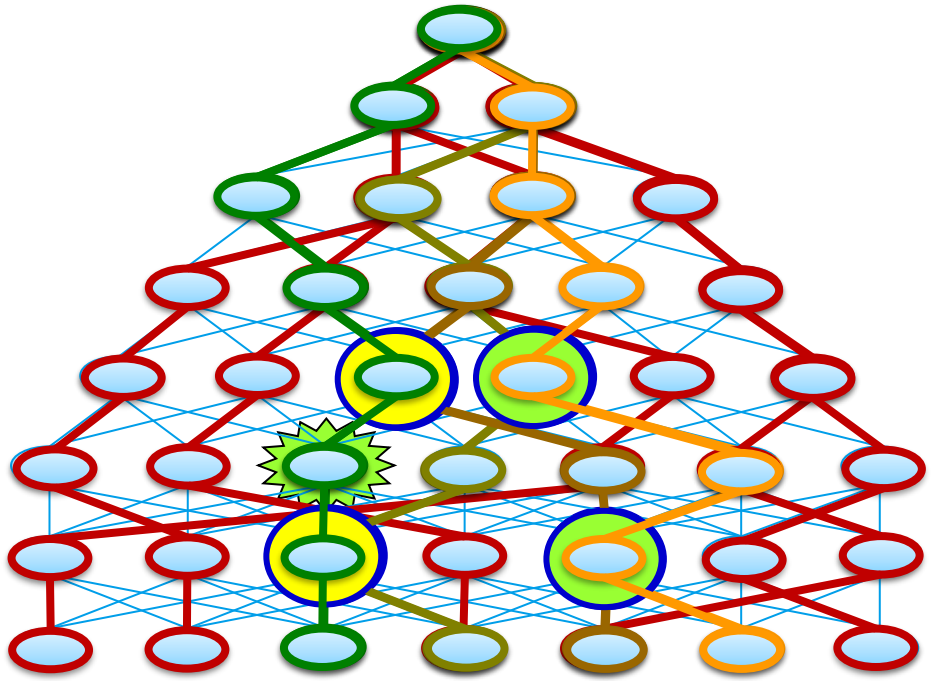
- Design: DMA Engine
  - Bus interface
  - Two memory interfaces
  - Eight DMA channels



- Testbench: UVM
  - Sequence to control transfers
    - Sequence item specifies transfer
    - Programs DMA registers
    - Waits for end-of-transfer interrupt
  - Functional coverage on transfer descriptors
  - Scoreboard checks results

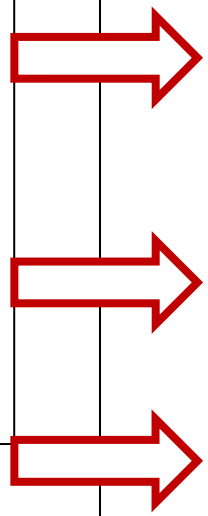
Setting Coverage Goals in UVM

```
class wb_dma_descriptor extends uvm_sequence_item;
`uvm_object_utils(wb_dma_descriptor)

  rand bit[5:0] channel;
  rand bit      mode;
  rand bit      inc_src;
  rand bit      inc_dst;
  rand bit      src_sel;
  rand bit      dst_sel;

  bit[31:0]      src_addr;
  bit[31:0]      dst_addr;
  rand bit[11:0] tot_sz;
  rand bit[2:0]  trn_sz;
  rand bit[8:0]  chk_sz;
  rand bit[31:0] transfer_sz;
  ...
endclass
```



Setting Coverage Goals in UVM

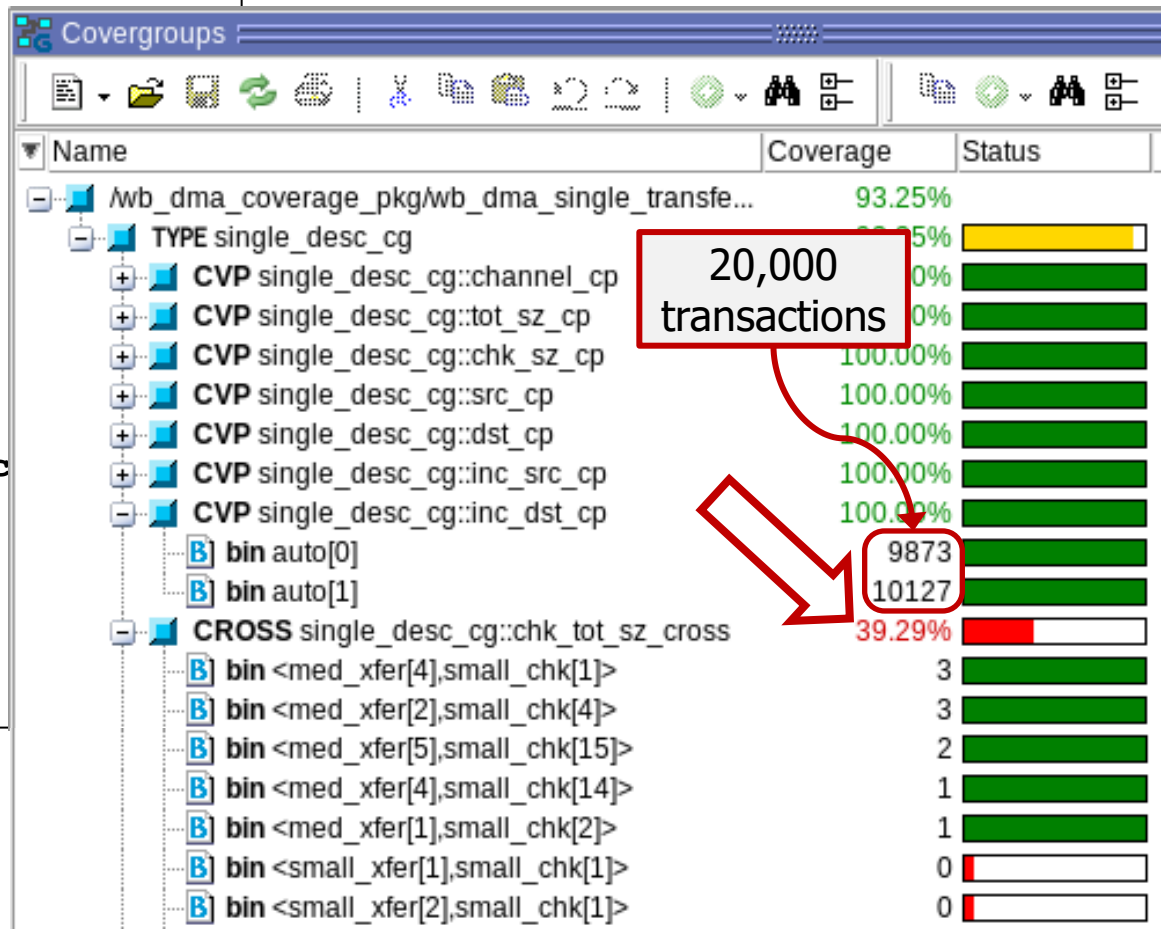
```
class wb_dma_descriptor extends uvm_sequence_item;
`uvm_object_utils(wb_dma_descriptor)

  rand bit[5:0] channel;
  rand bit      mode;
  rand bit      inc_src;
  rand bit      inc_dst;
  rand bit      src_sel;
  rand bit      dst_sel;

  bit[31:0]      src_addr;
  bit[31:0]      dst_addr;
  rand bit[11:0] tot_sz;
  rand bit[2:0]  trn_sz;
  rand bit[8:0]  chk_sz;
  rand bit[31:0] transfer_sz;
  ...
endclass
```



```
class wb_dma_single_transfer_descriptor_cov
      extends uvm_subscriber #(wb_dma_descriptor);
  ...
  wb_dma_descriptor                          desc;

  covergroup single_desc_cg;
    channel_cp : coverpoint desc.channel {
      bins channels[] = {[0:30]};
    }
    tot_sz_cp : coverpoint desc.tot_sz {
                  bins small_xfer[] = {[1:16]};
                  bins med_xfer[16] = {[16:4089]};
                  bins huge_xfer[] = {[4090:4095]};
    }
    chk_sz_cp : coverpoint desc.chk_sz {
                  bins small_chk[] = {[1:15]};
                  bins large_chk[] = {[16:256]};
    }
    chk_tot_sz_cross : cross tot_sz_cp, chk_sz_cp;
    ...
  endgroup
  ...
endclass
```

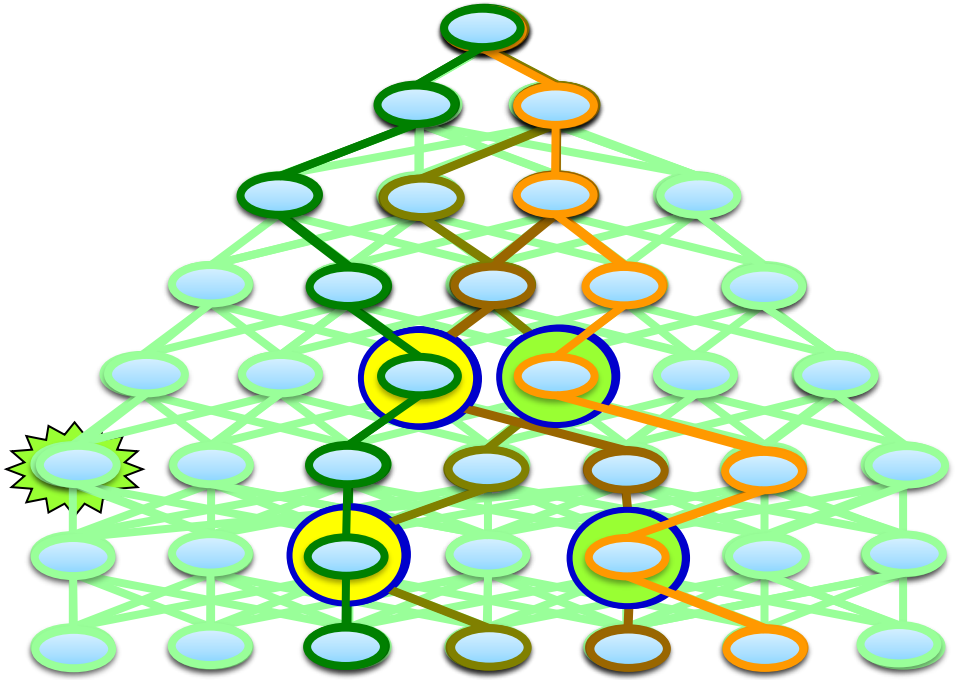
UVM Constrained-Random Sequence

```
class wb_dma_rand_single_transfer_seq extends wb_dma_transfer_seq;
`uvm_object_utils(wb_dma_rand_single_transfer_seq)

  virtual task body();
    wb_dma_descriptor desc;

    repeat (200) begin
      desc = wb_dma_descriptor::type_id::create("desc");

      start_item(desc);
      if (!desc.randomize()) begin
        `uvm_fatal(get_name(), "Failed to randomize sequenc
      end
      finish_item(desc);
    end
  endtask
endclass
```



inFact Portable Stimulus and Functional Coverage

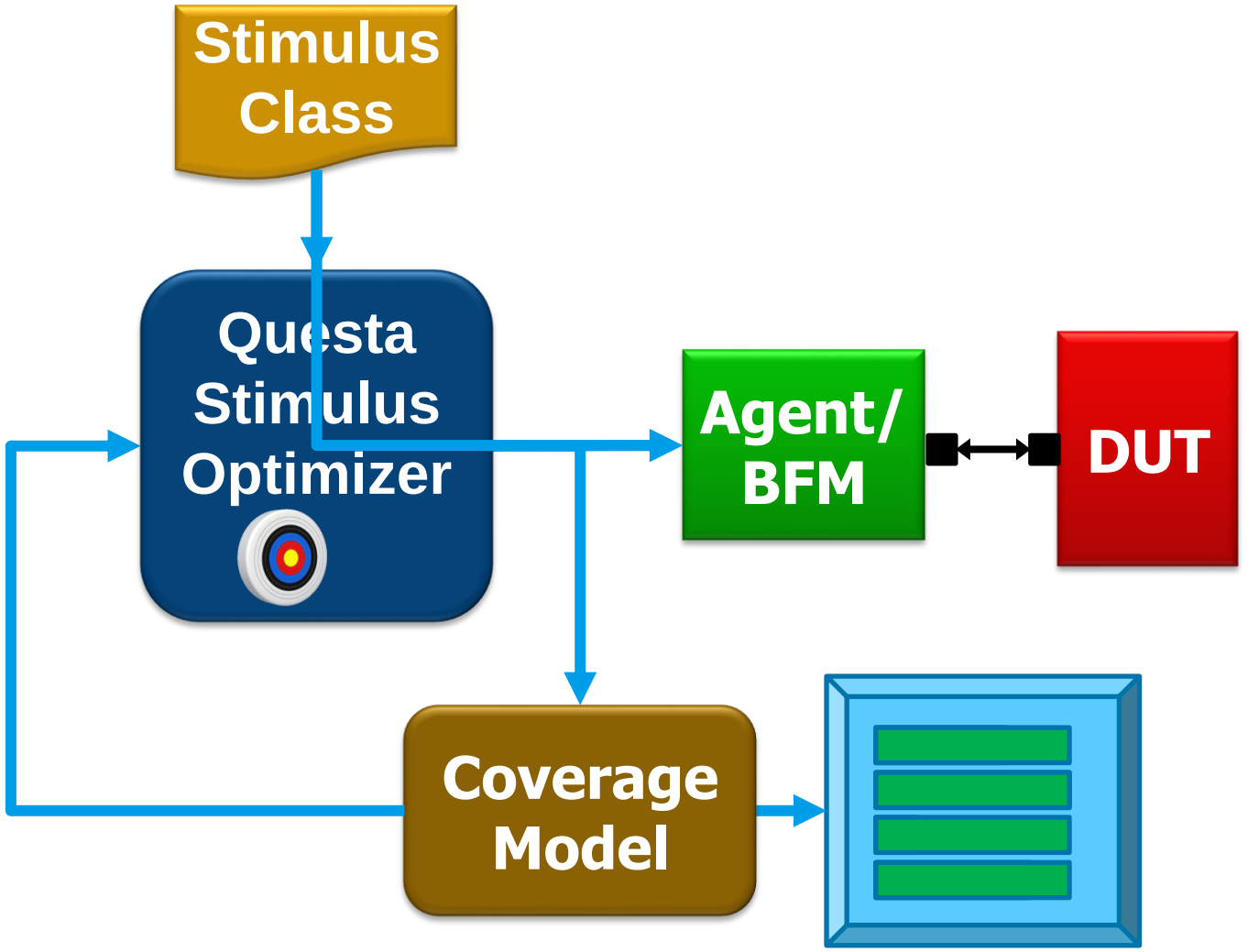
Active Coverage



- Functional Coverage defines critical states
- inFact can target these states first
- Maximum coverage in the fewest tests
- Additional tests can go beyond for bug hunting

Closing Coverage with UVM Apps

- Start with Your Existing UVM Definitions
- Automatically Target Coverage
- Analyze Model to Create Additional Coverage Targets
  - Find corner cases that you didn't think of



Importing UVM into inFact for Portable Stimulus

```
qso-run :
  $(Q)rm -rf qso
  $(Q)mkdir qso
  $(Q)$(INFACT_HOME)/bin/qso \
      wb_dma_env_pkg::wb_dma_descriptor -unit wb_dma_coverage_pkg \
      -covergroup wb_dma_coverage_pkg::wb_dma_single_transfer_descriptor_cov::single_desc_cg \
      -name wb_dma_descriptor_gen -o qso/wb_dma_d                    svh
```

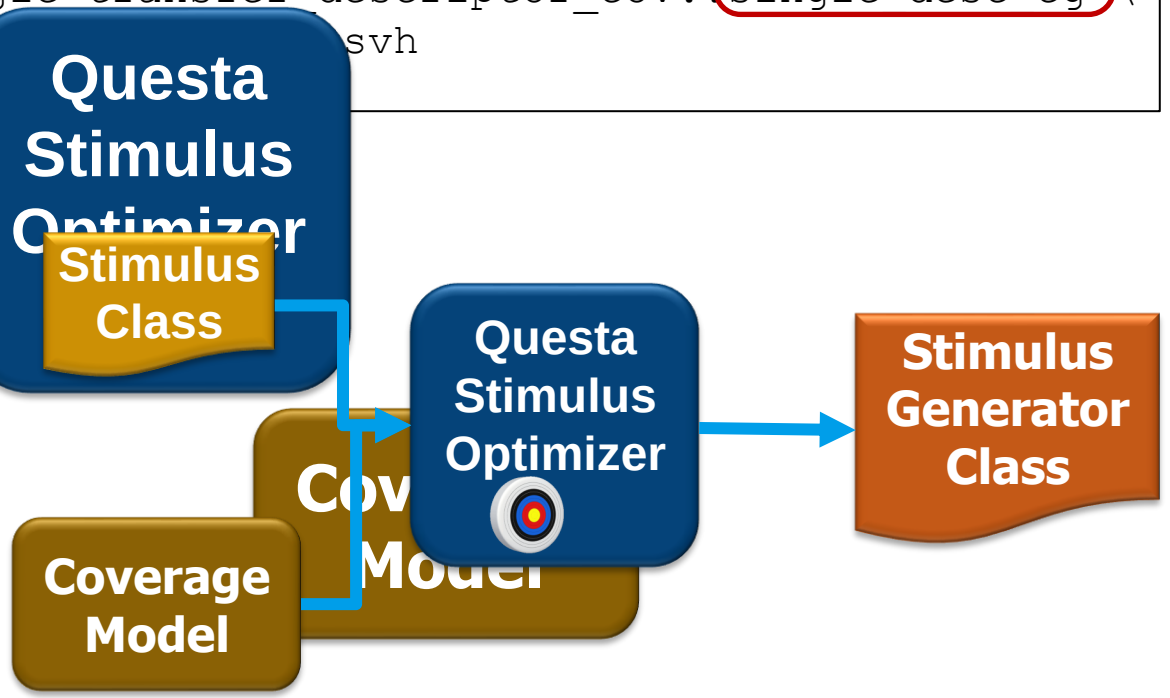
Stimulus Class

ransaction type

The covergroup

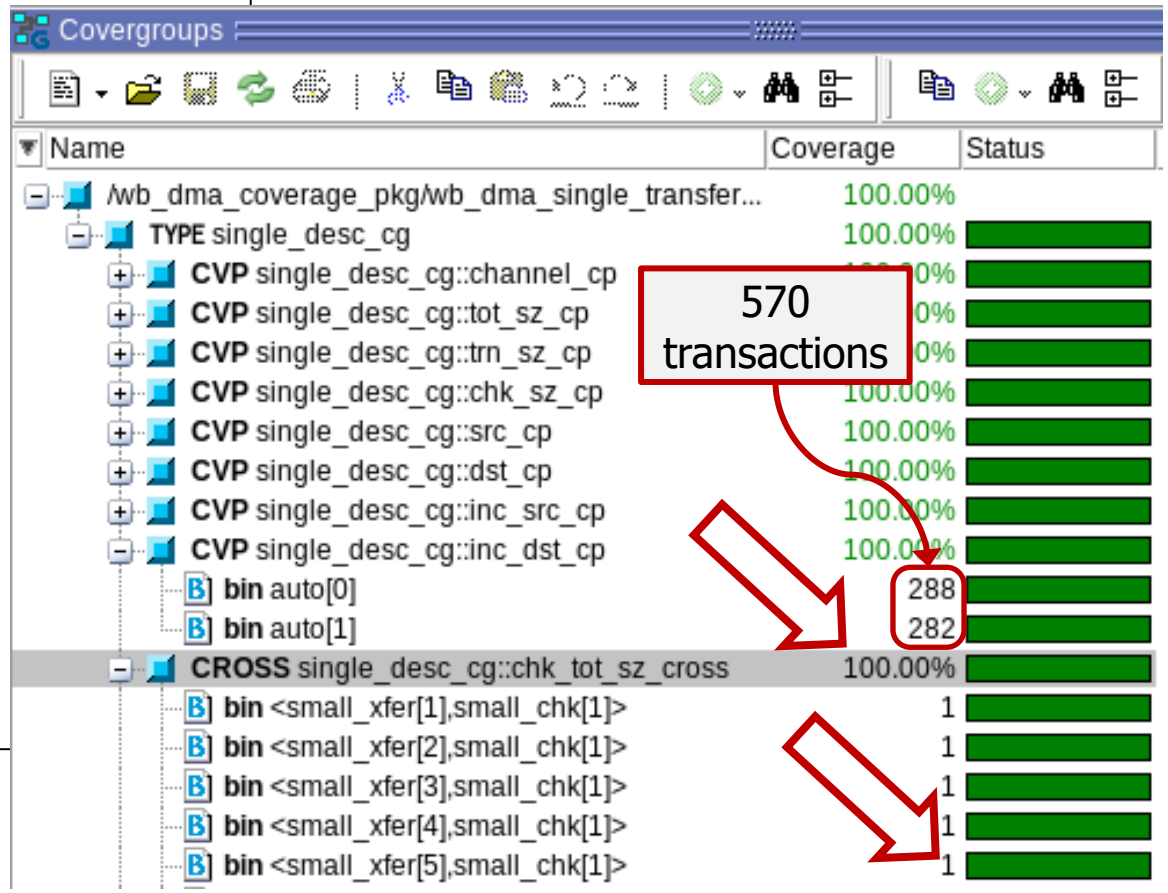
The in

- inFact reads SV classes and covergroup
  - Creates a portable stimulus model
  - Creates a stimulus-generator class
  - Replaces the call to randomize
- inFact stimulus follows same constraints
  - Achieves coverage more quickly
  - Bug-hunt with expanded coverage goals



inFact Portable Stimulus Sequence

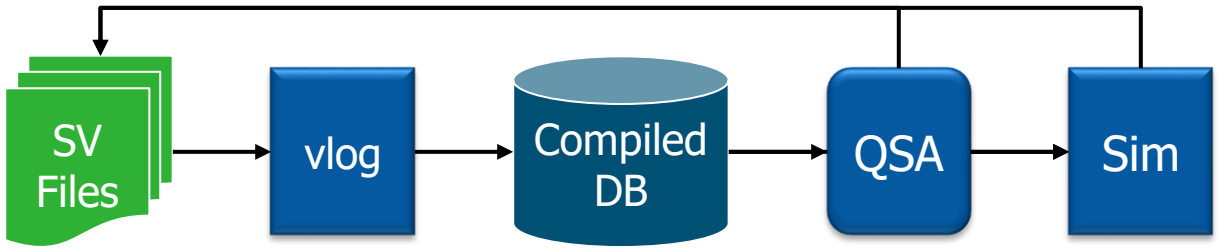
```
class wb_dma_infact_single_transfer_seq extends wb_dma_transfer_seq;
`uvm_object_utils(wb_dma_infact_single_transfer_seq)

  virtual task body();
    wb_dma_descriptor desc;
    wb_dma_descriptor_gen infact_gen;

    infact_gen = new({get_full_name(), ".infact_gen"});

    while (!infact_gen.allCoverageGoalsMet()) begin
      desc = wb_dma_descriptor::type_id::create("desc");

      start_item(desc);

      infact_gen.ifc_fill(desc);

      finish_item(desc);
    end
  endtask
endclass
```

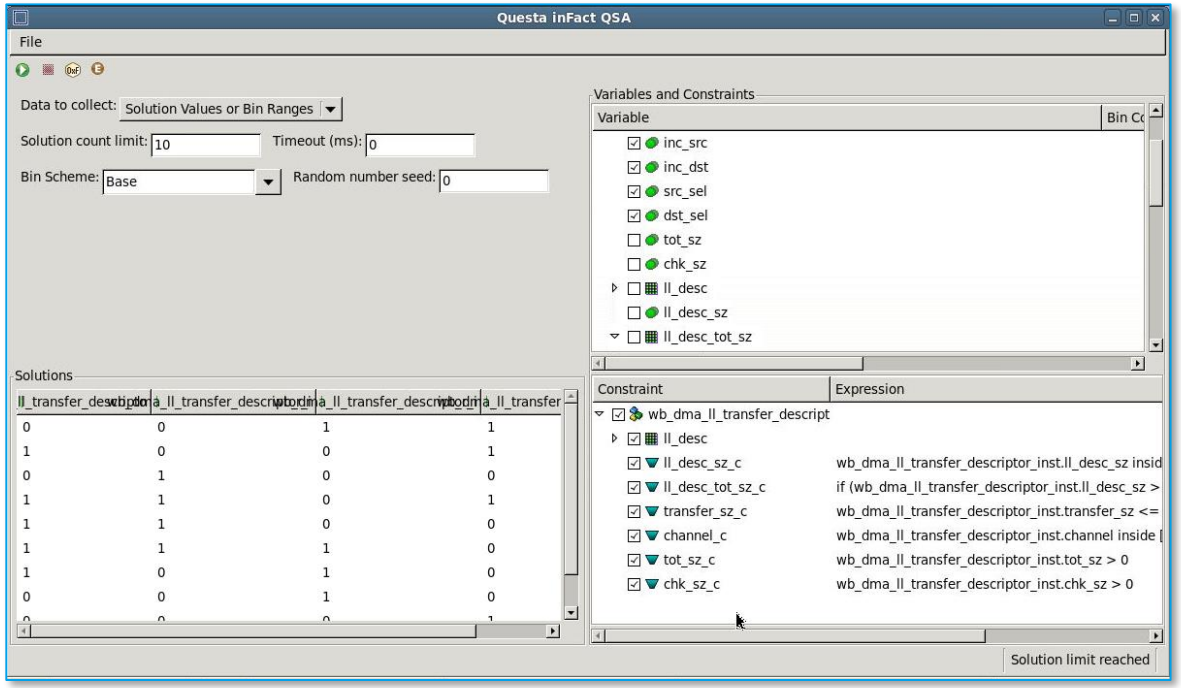


Questa Stimulus Analyzer

Automation Reduces Stimulus-Development Time

- Creating the right constraints is difficult!
  - Over-constrained stimulus misses key cases
  - Under-constrained stimulus creates illegal cases
- Debugging constraints is time-consuming!
  - Compile, elaborate, simulate.
  - Run simulation and see what happens
  - Each time around the loop takes minutes
- Portable stimulus is statically analyzable
- Constraint Explorer short-circuits the debug loop
  - Check for solvability
  - Check for key cases
  - Shortens the loop to seconds
- Go to simulation with debugged constraints

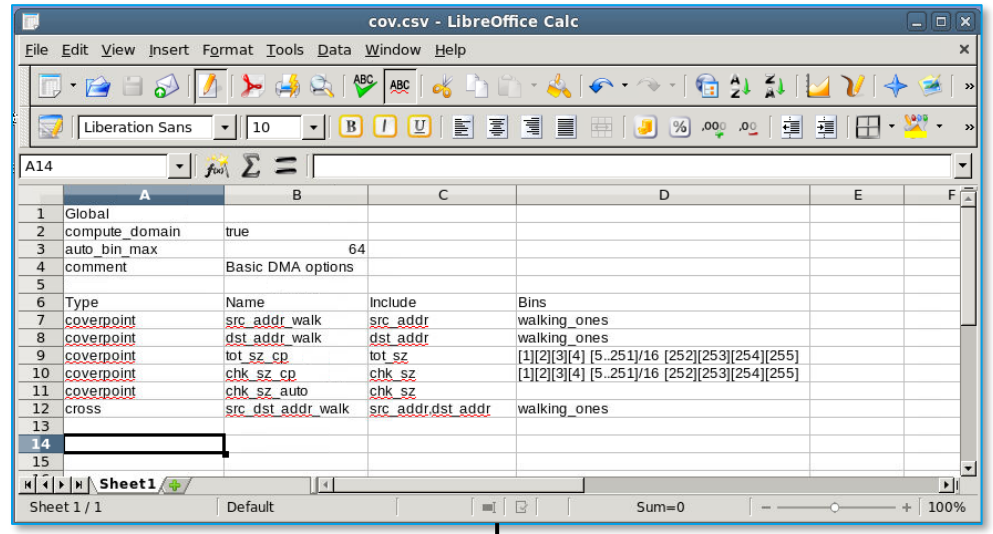




Questa Coverage Creator

Automated Correct by Construction Covergroups

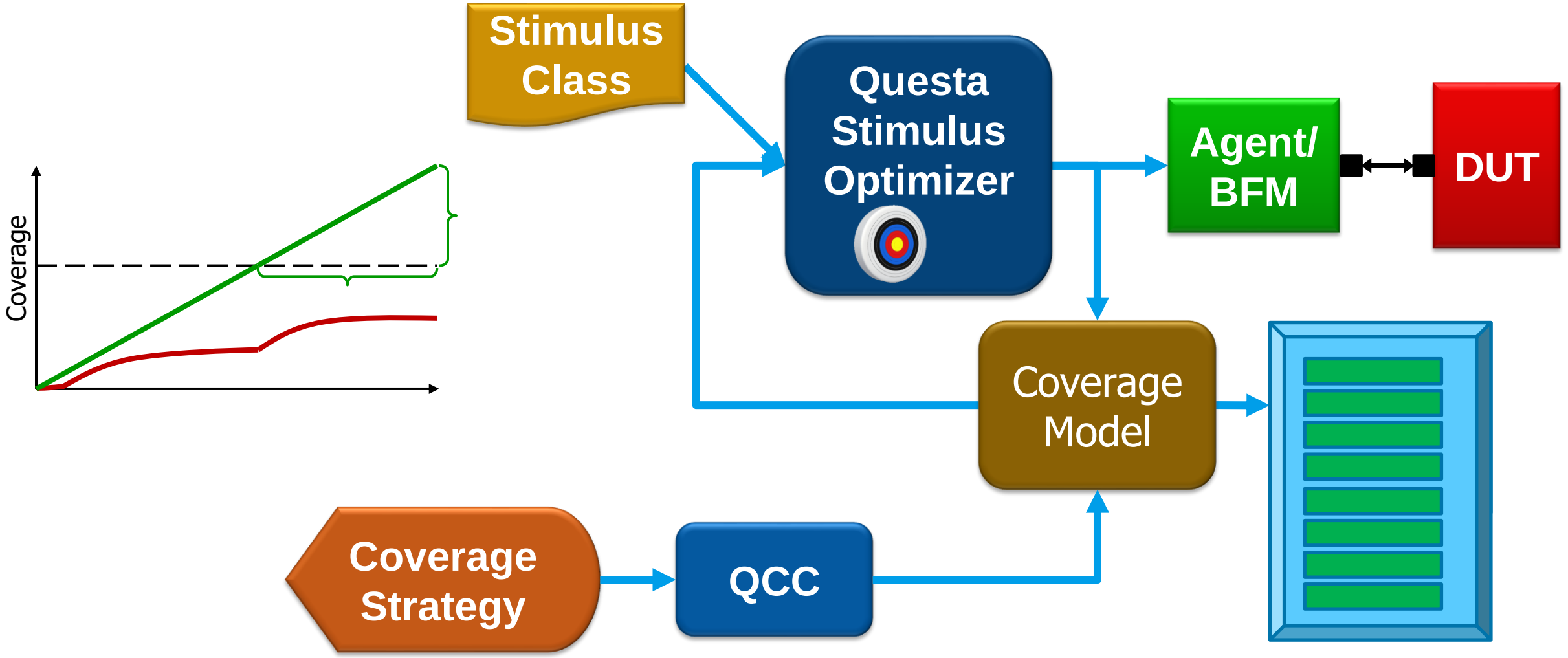
- Questa Coverage Creator automates coverage creation
  - Removes grunt-work from creating explicit functional coverage
  - Auto-creates coverage goals based on class fields and constraints
  - Creates SystemVerilog source code
- Provides earlier visibility into the verification process
  - Effectiveness of stimulus
  - Confirms design features exercised
- Correct-by-construction covergroups
  - Easily re-generate when testbench changes
  - Reduces post-regression coverage hole debug



	A	B	C	D	E
1	Global				
2	compute_domain	true			
3	auto_bin_max	64			
4	comment	Basic DMA options			
5					
6	Type	Name	Include	Bins	
7	coverpoint	src_addr_walk	src_addr	walking_ones	
8	coverpoint	dst_addr_walk	dst_addr	walking_ones	
9	coverpoint	tot_sz_cp	tot_sz	[1][2][3][4] [5..251]/16 [252][253][254][255]	
10	coverpoint	chk_sz_cp	chk_sz	[1][2][3][4] [5..251]/16 [252][253][254][255]	
11	coverpoint	chk_sz_auto	chk_sz		
12	cross	src_dst_addr_walk	src_addr,dst_addr	walking_ones	
13					
14					
15					

```
covergroup infact_pkt_gen_cov_cg;
    option.name = name;
    option.per_instance = 1;
    // Path Coverage pkt_cross
    pkt_cross_pkt_inst_flags_cp : coverpoint this.m_pkt_inst.flags {
        option.weight = 0;
        bins pkt_inst_flags[] = {[64'd0:64'd1]};
    }
    pkt_cross_pkt_inst_type_cp : coverpoint infact_pkt_gen_enums_c::pkt__type'(this.m_pkt_inst.type) {
        option.weight = 0;
        bins pkt_inst_type[] = {[infact_pkt_gen_enums_c::TP0:infact_pkt_gen_enums_c::TP3]};
    }
    pkt_cross_pkt_inst_d_size_cp : coverpoint this.m_pkt_inst.d_size {
        option.weight = 0;
        bins pkt_inst_d_size[] = {64'd1, 64'd255};
        bins pkt_inst_d_size_1 = {[64'd2:64'd26]};
        bins pkt_inst_d_size_2 = {[64'd27:64'd51]};
```

Expand Coverage for Bug Hunting



You Focus on *What*, Questa inFact Handles *How*

- PSS Apps let the tool do the work
  - Questa Stimulus Analyzer
  - Questa Coverage Creator
  - Questa Stimulus Optimizer
- Enhance your constrained-random or directed tests to take advantage of scenario-level randomization
- Let inFact explore all possible options
- Automatically target coverage goals
  - Expand coverage goals for bug hunting


```
class wb_dma_infact_single_transfer_seq extends wb_dma_transfer_seq;
    `uvm_object_utils(wb_dma_infact_single_transfer_seq)


   /**
    * Task: body
    *
    * Override from class
    */
   virtual task body();
       wb_dma_descriptor_single_desc_cov_gen cov_gen = new(
               {get_full_name(), ".cov_stim_gen"});
       wb_dma_single_transfer_descriptor desc;

       while (!cov_gen.allCoverageGoalsHaveBeenMet()) begin
                                                  or::type_id::create("desc");
       start_item(desc);
       cov_gen.ifc_fill(desc);
       finish_item(desc);


endclass
```

inFact-Enabled UVM Sequence

Mentor® A Siemens Business

Who should consider to apply the Questa inFact?

- Who want to achieve an early coverage closure
- Who want to obtain broad coverage to maximize early bug detection
- Who don’t want to spend too much time to tweak constraints for coverage closure
- Who want to verify constraints before simulation
- Who want to use a scenario test model in PSS
- Who use any simulator for testing
  - Supports 3rd party simulators

A Siemens Business

www.mentor.com