2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Formal Verification Tutorial – Breaking Through the Knowledge Barrier

Sean Safarpour - Synopsys, Inc.

Iain Singleton - Synopsys, Inc.

Shaun Feng - Samsung Austin R&D Center

Syed Suhaib - Nvidia Corp.

Mandar Munishwar - Qualcomm, Inc.

SAMSUNG NVIDIA QUALCOMM® SYNOPSYS®

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

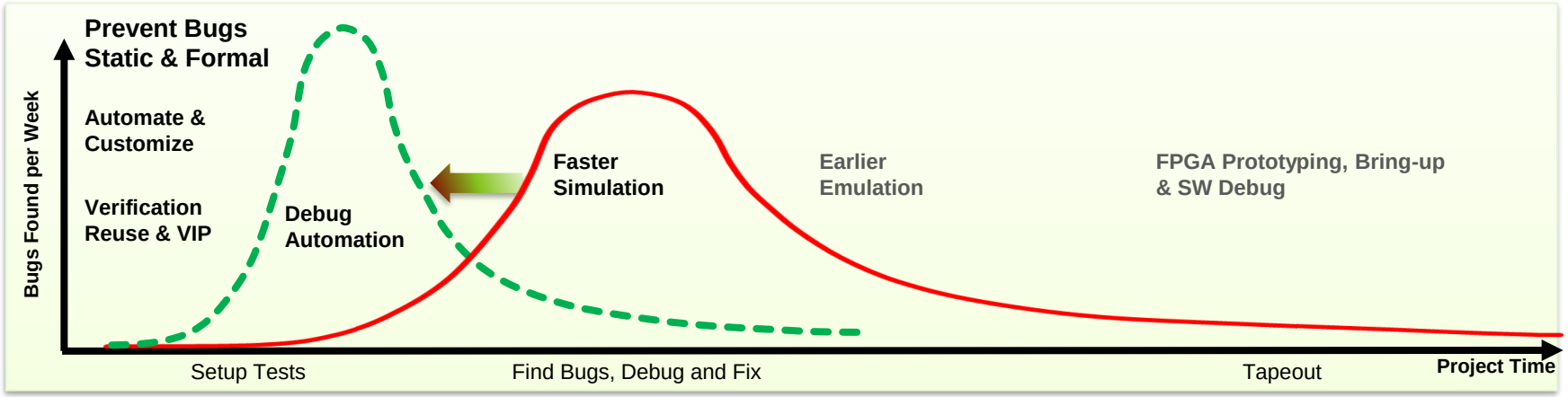
• Introduction	: Sean Safarpour	(20 min)
• Induction & Invariants – Steps to Convergence	: Iain Singleton	(45 min)
• Efficient Formal Modeling Techniques	: Shaun Feng	(45 min)
Break		(10 min)
• Architectural Formal Verification for Coherency	: Syed Suhaib	(45 min)
• Formal Sign-off	: Mandar Munishwar	(45 min)

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Introduction

Sean Safarpour, Synopsys
Email: seans@synopsys.com

Formal Key Enabler for "Shift Left"



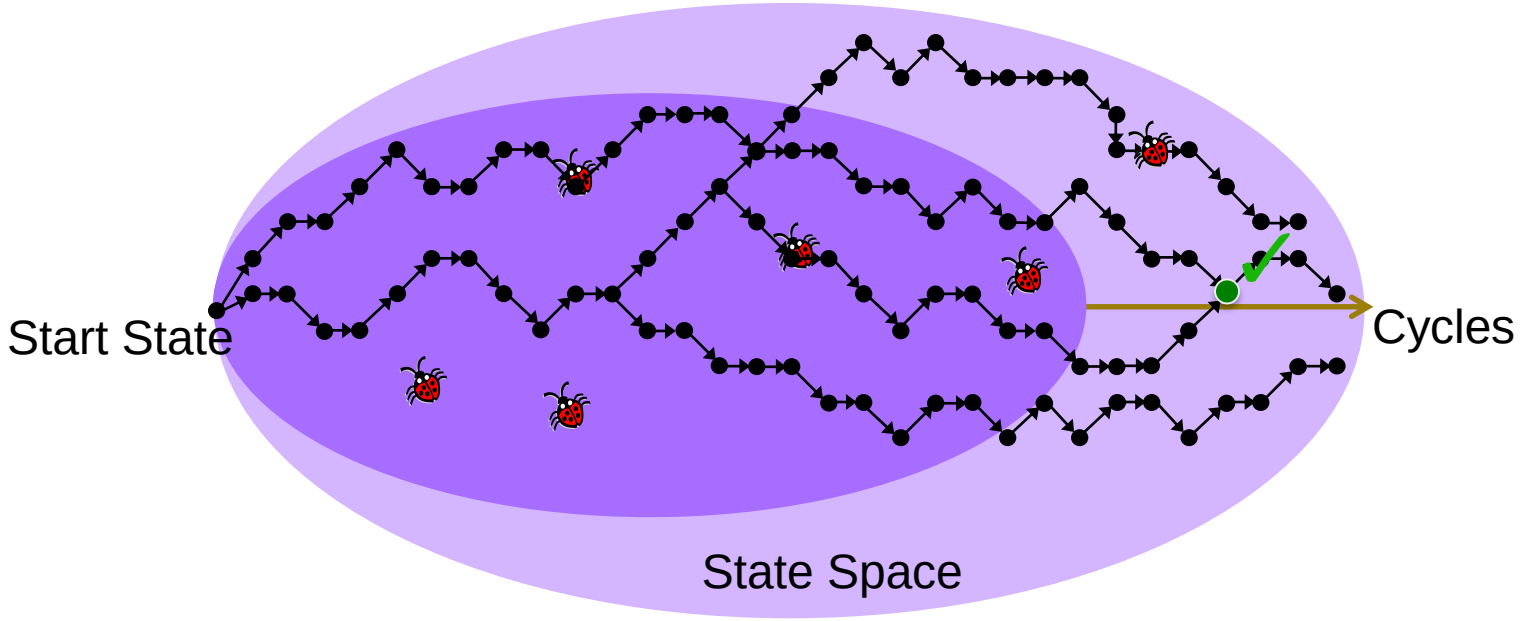
The Verification Questions Remain:

- How can we reduce our overall verification time?
- How can we improve efficiency?
- How can we find the late bugs earlier?
- How can we prevent bugs slipping into Silicon?

How to Improve Verification Confidence

- Simulation cycles aren't scaling
  - Need to look at each problem differently
- Let's break down the verification problem
  - Verification plan consists of individual tasks
  - Some well suited for simulation
  - Some well suited for emulation
  - Some well suited for static/formal verification
  - Use the right task for the right problem
- Why consider multiple tools in the verification flow?
  - Not all problems can be solved by the same approach
  - Use the right tool for the right problem
    - Find bugs, saves time and $$$

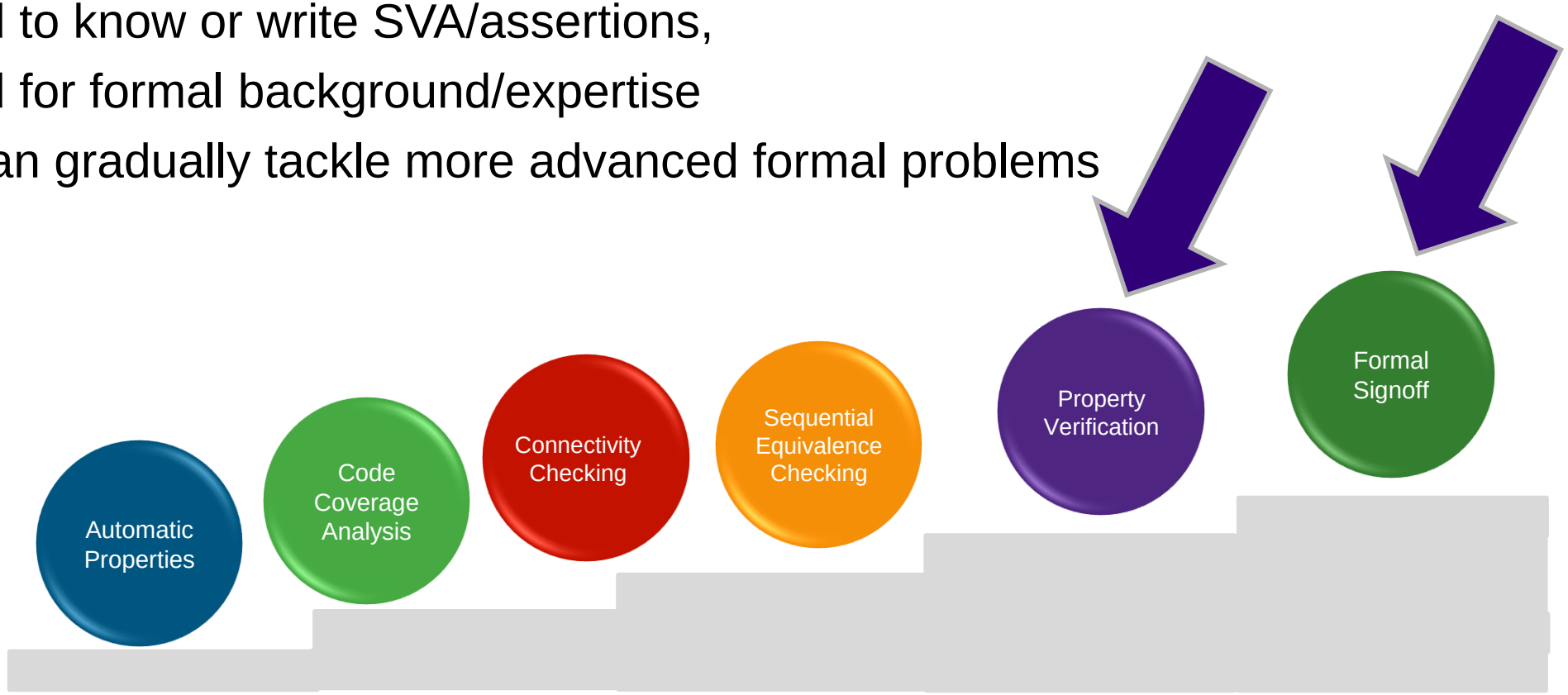
Complementing Simulation with Formal



- Formal explores design state space exhaustively
- Simulation is not exhaustive but explores deep sequential behavior

Stairway to Formal

- Formal Applications (Apps) solve specific problems very well
- Easy to setup & use & debug
- No need to know or write SVA/assertions,
- No need for formal background/expertise
- Users can gradually tackle more advanced formal problems

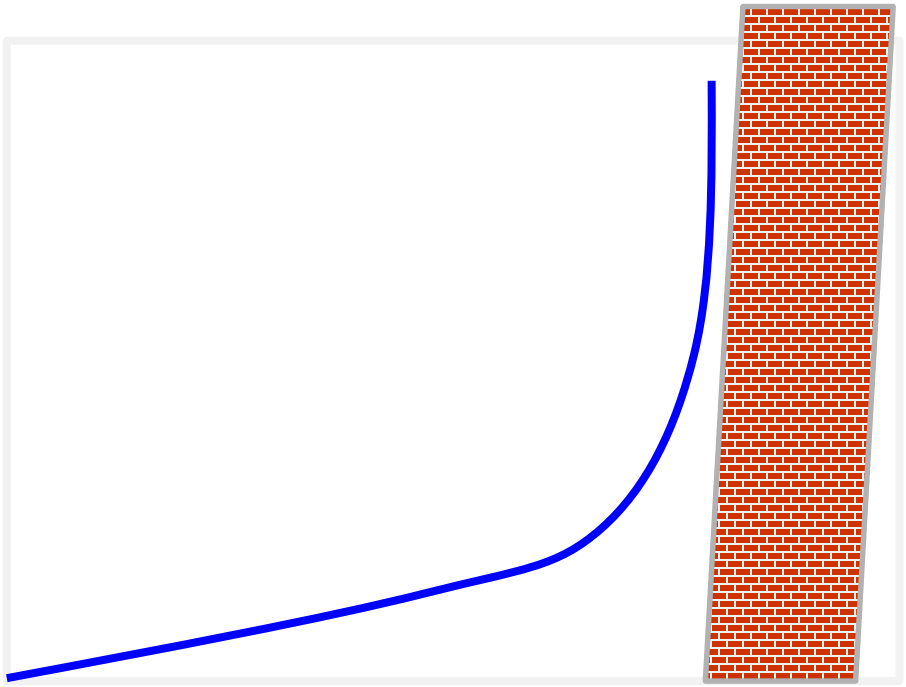


Property Verification

- Formal Property Verification
  - Very powerful
  - Very flexible: can be deployed on many problems
  - Size limited: block/IP level
  - Limited to Control Paths
  - Exponential problem: no conclusive answer

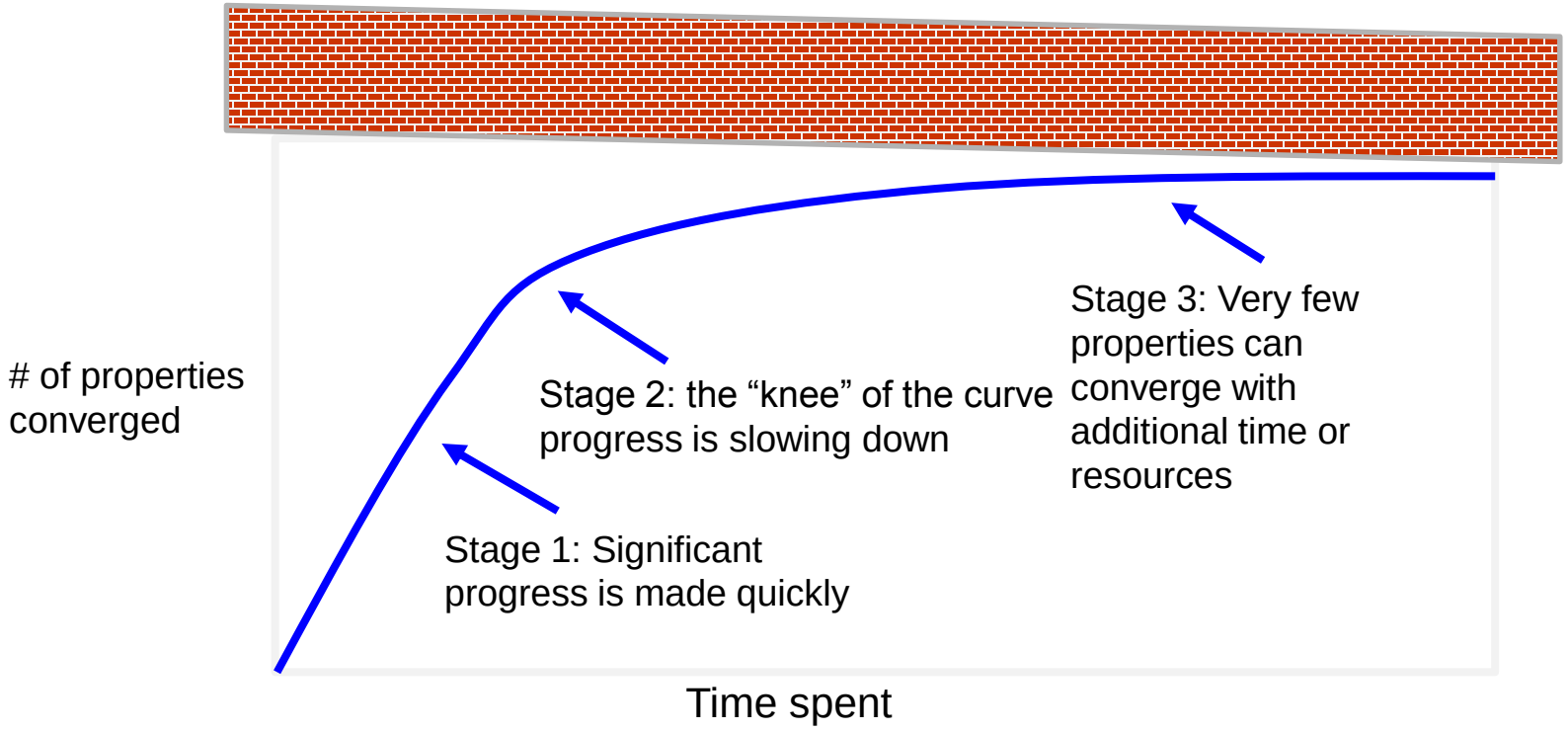
Nature of the Problem

- Formal Verification problem is exponential in nature
- For hard problems, at some point progress stop



Nature of the Problem

- Plateau graph:
  - When looking at a large number of properties over time …progress appears to stops



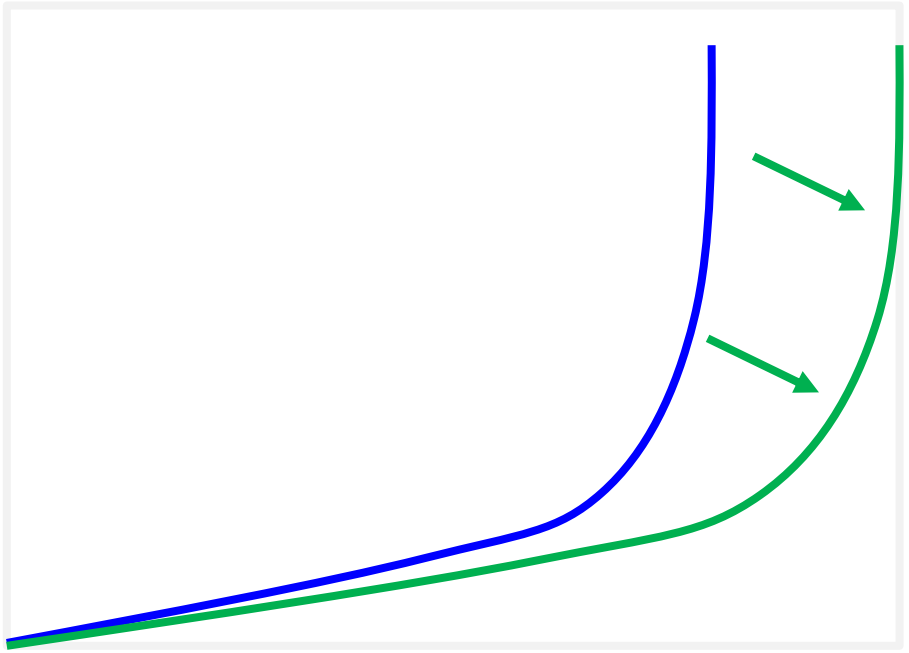
Property Verification

- Formal Property Verification
  - Very powerful
  - Very flexible: can be deployed on many problems
  - Size limited: block/IP level
  - Limited to Control Paths
  - Exponential problem: no conclusive answer

- But in the hands of an expert…
  - Size limit can be worked around
  - Datapath can be handled
  - Exponential effects can be managed

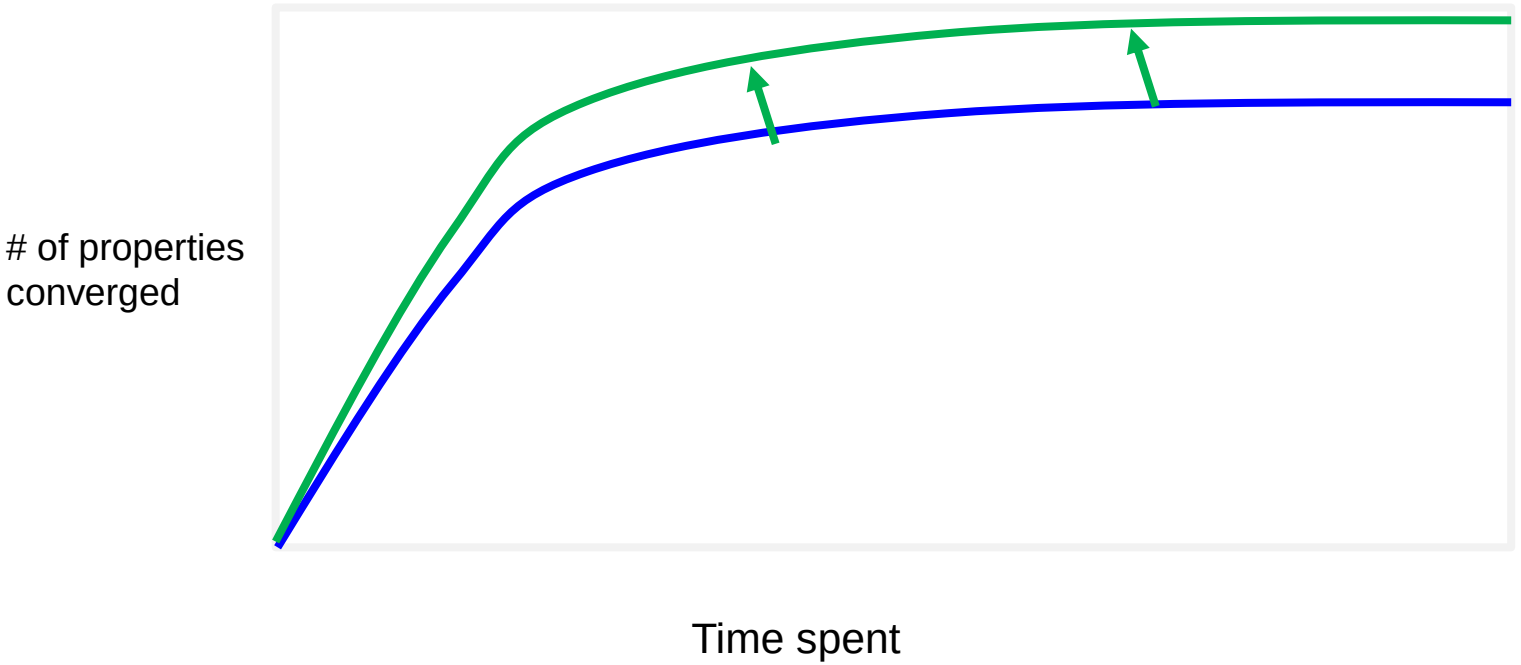
Nature of the Problem

- With some “tricks” / know-how we can jump to another curve
- Might be good enough to solve our problem



Nature of the Problem

- Plateau graph
  - More properties will converge for the same amount of time



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Secrets Disclosed

- Through Years of experience and countless projects…
- Formal expert have discovered and mastered techniques to go beyond limitations
- Presenters will disclose some of their secrets…
  - Abstraction techniques
  - Symbolic Variables
  - Invariants and Induction
  - Architectural level checks
  - Signing off with formal

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q&A

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

• Introduction	: Sean Safarpour	(20 min)
• Induction & Invariants – Steps to Convergence	: Iain Singleton	(45 min)
• Efficient Formal Modeling Techniques	: Shaun Feng	(45 min)
Break		(10 min)
• Architectural Formal Verification for Coherency	: Syed Suhaib	(45 min)
• Formal Sign-off	: Mandar Munishwar	(45 min)

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

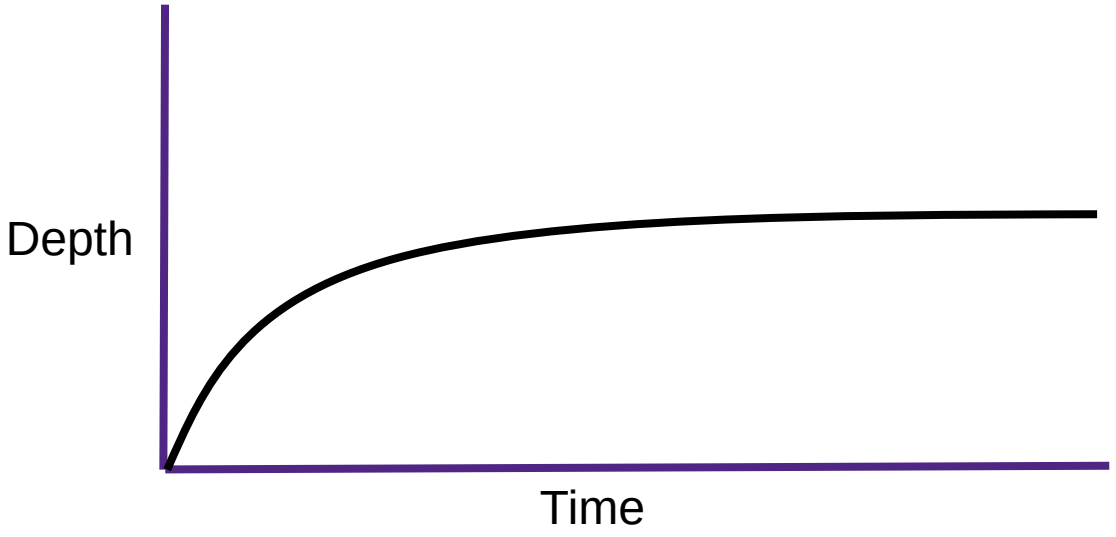
Induction & Invariants – Key Steps to Convergence

Iain Singleton - Synopsys, Inc.

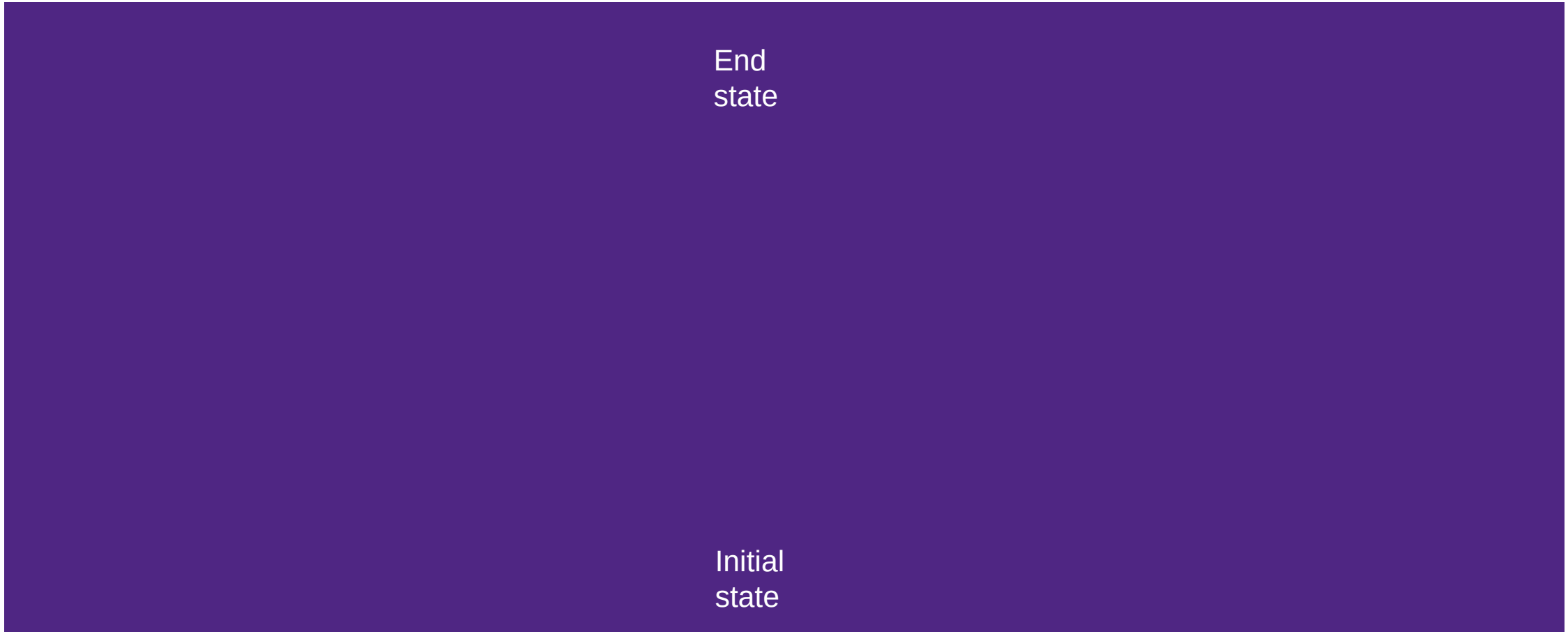
Email: isingle@synopsys.com

The Formal Convergence Problem

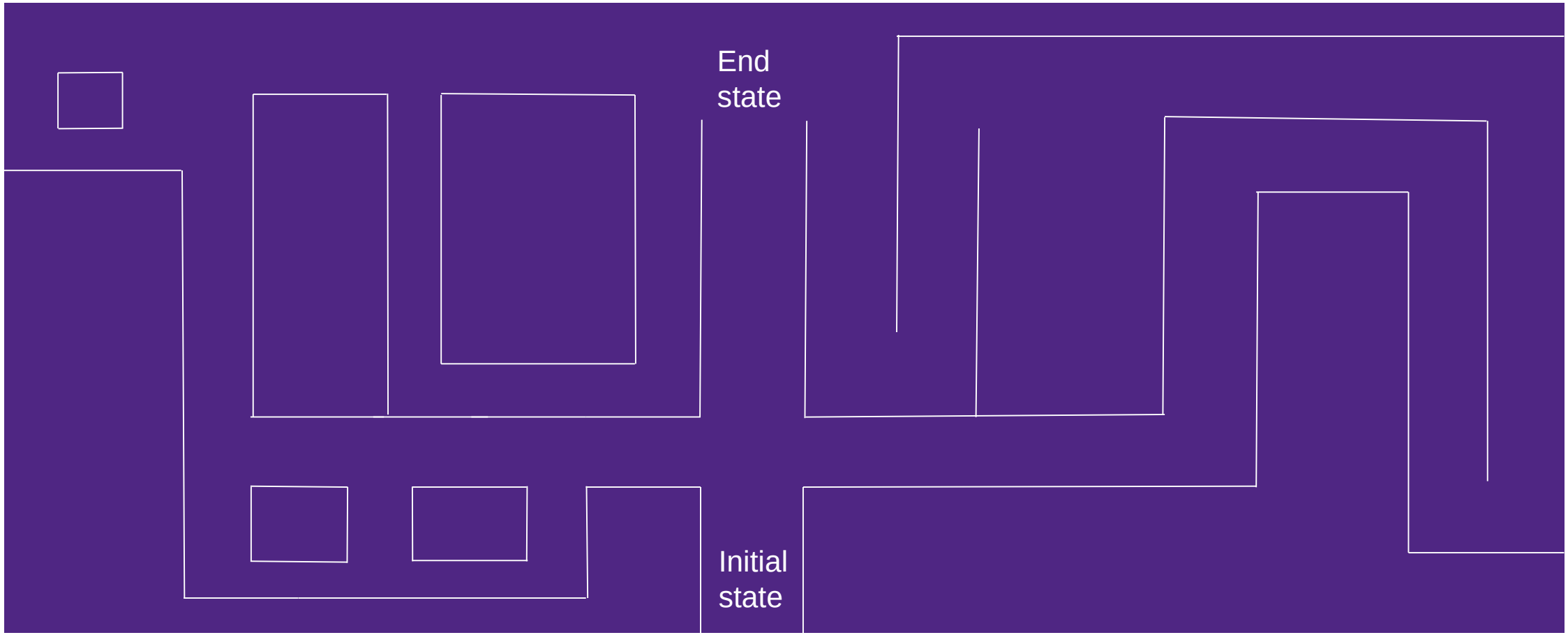
- The Satisfiability (SAT) problem:
  - Given a Boolean expression is there a set of values which will evaluate the expression to true
  - For each variable, *n*, there are 2^n possible values which must be tested
  - This creates an exponential NP-complete problem



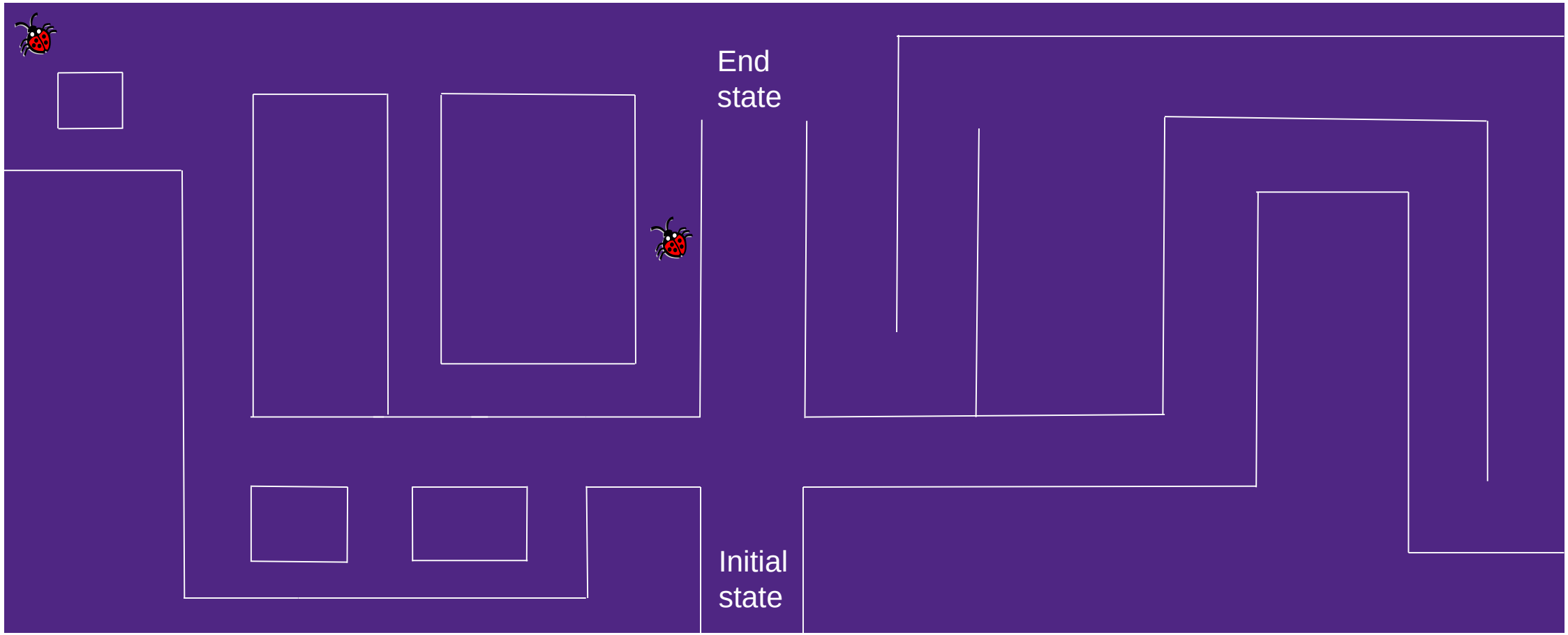
State Space Exploration



State Space Exploration



State Space Exploration



Improving Convergence

- Not all problems are created equal
  - Complexity is impacted by a number of factors

- Overcoming complexity is one of formals biggest challenges
  - Abstractions
  - Property Decomposition
  - Divide and Conquer
  - Case Splitting
  - **Invariants**
  - **Induction**

Invariants

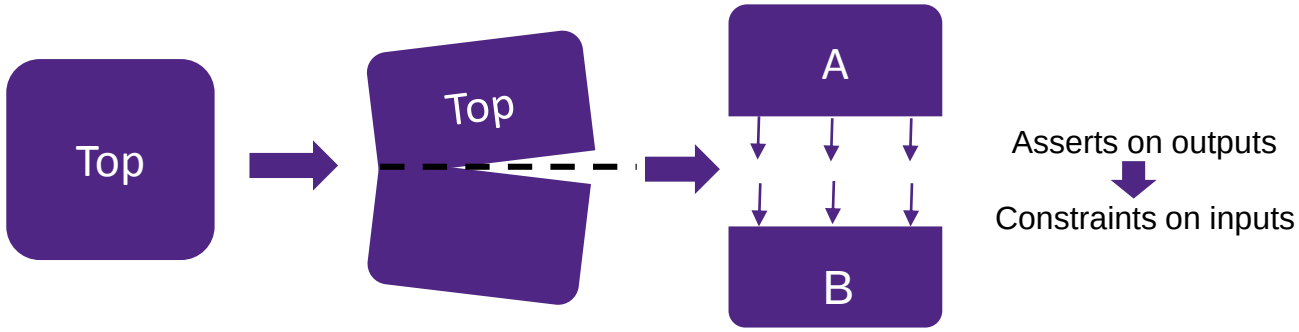
- What is an invariant?

A function, quantity or property which remains unchanged when a specific transformation is applied

- All assertions can be invariants

- Some invariants can improve convergence

Assume Guarantee

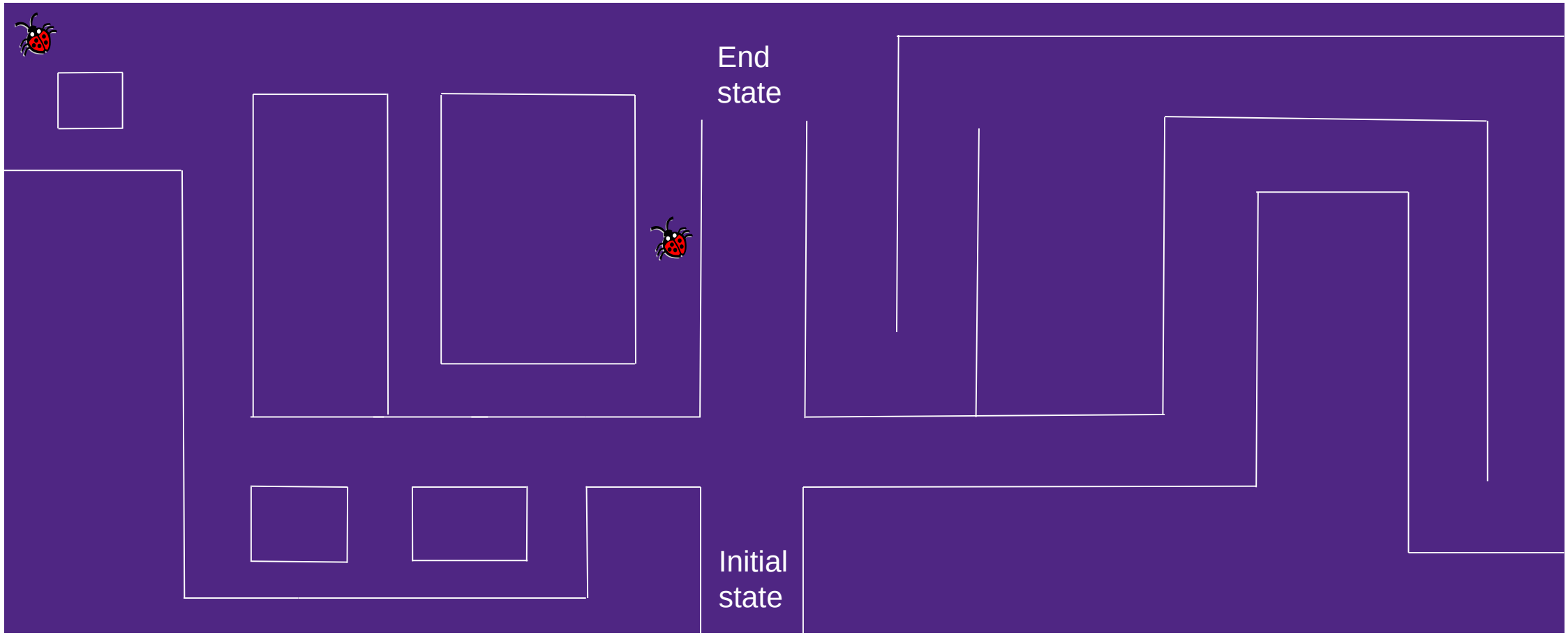


- Assume guarantee is used to guarantee constraints by proving them on the driving logic
- Used as a complement to divide and conquer methodology
- …but can also be used without splitting the design

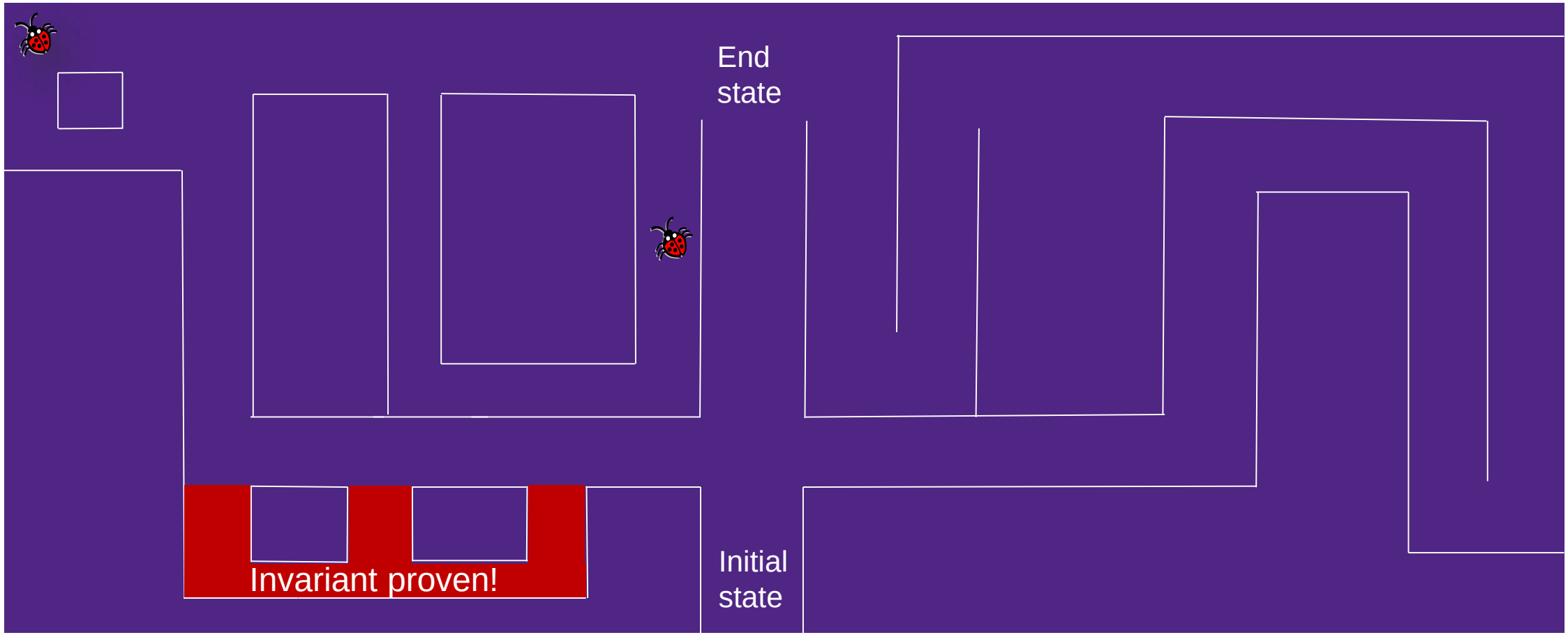
Invariants as Helper Properties

- Invariant properties can be used as constraints
  - Helps restrict state space
- Safe to assume a proven assertion
- State space reduction can help improve convergence time

State Space Exploration



State Space Exploration

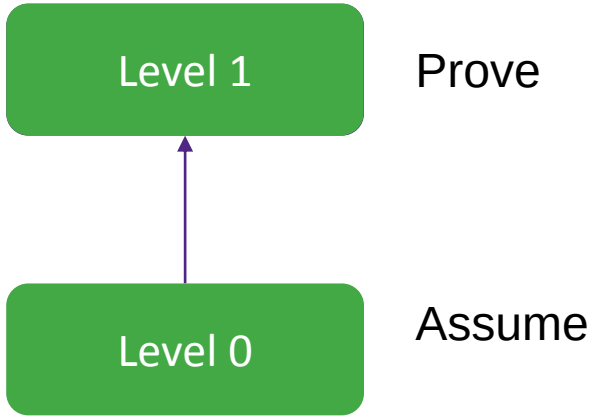


Invariants as Helper Properties

- Sometimes an invariant property will be almost as difficult to prove as the main property
- It may be necessary to chain invariants together in a prove-assume-prove flow
- Prove the easiest properties first, then slightly harder, then harder etc…

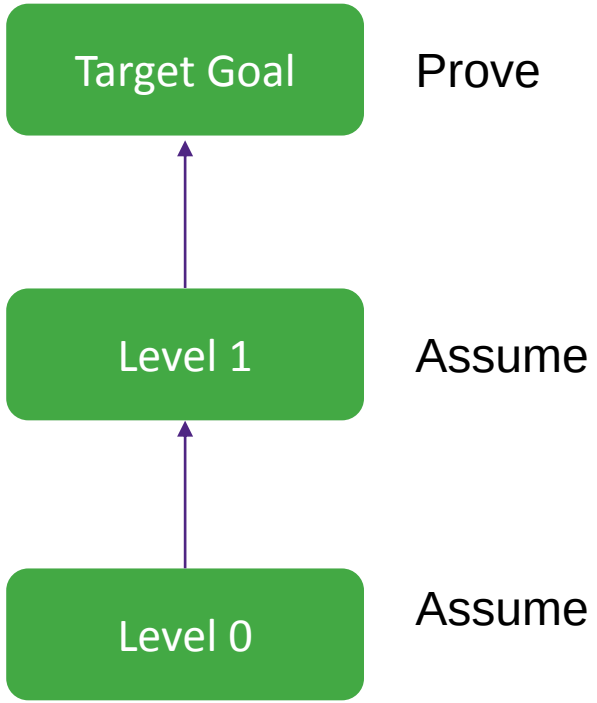
Invariants as Helper Properties

- Sometimes an invariant property will be almost as difficult to prove as the main property
- It may be necessary to chain invariants together in a prove-assume-prove flow
- Prove the easiest properties first, then slightly harder, then harder etc…

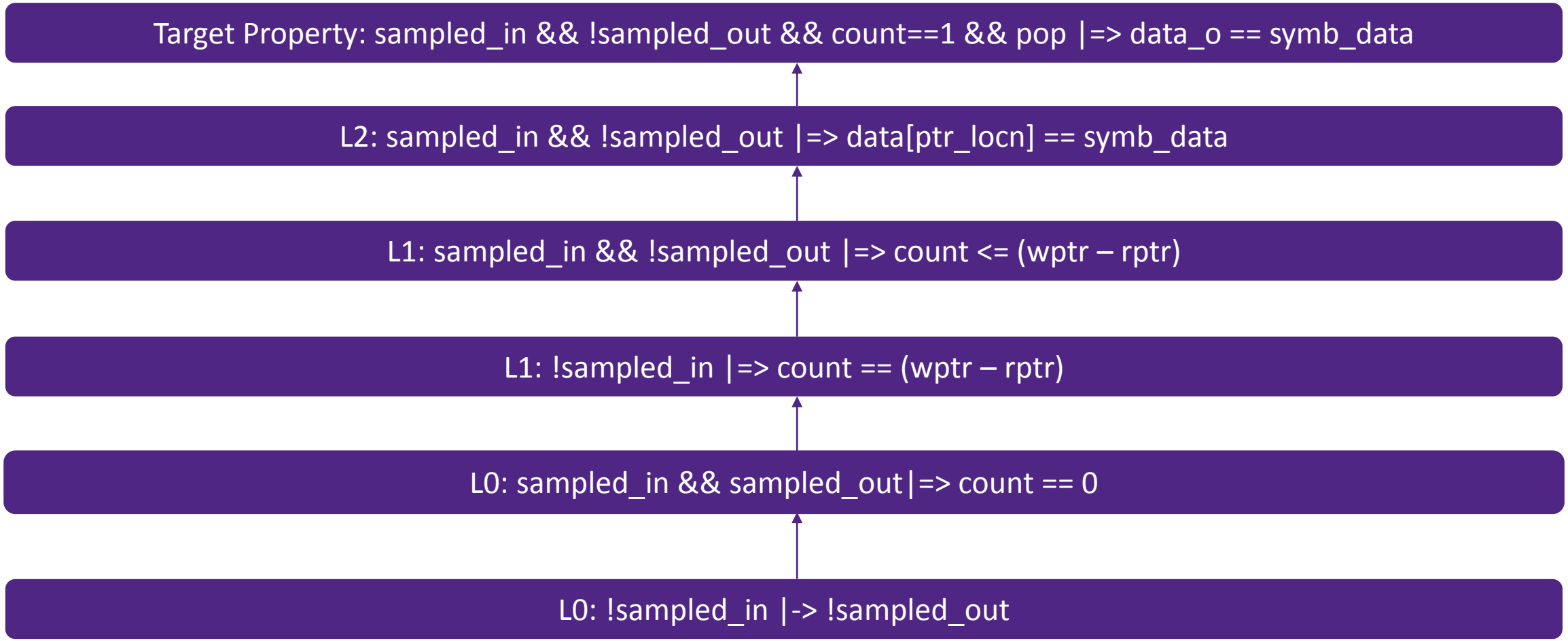


Invariants as Helper Properties

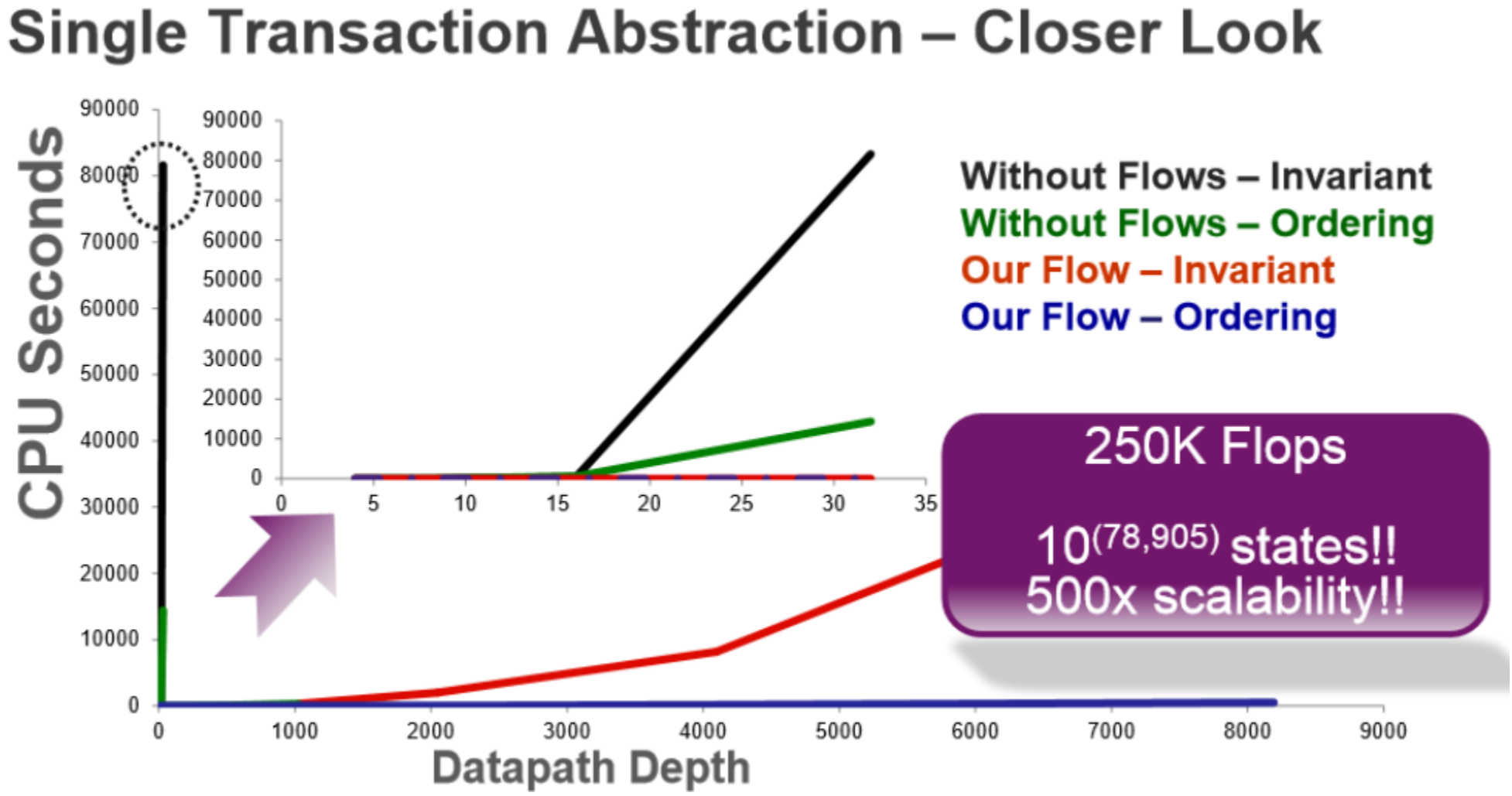
- Sometimes an invariant property will be almost as difficult to prove as the main property
- It may be necessary to chain invariants together in a prove-assume-prove flow
- Prove the easiest properties first, then slightly harder, then harder etc…



Invariants as Helper Properties – Example (DAC 2014)



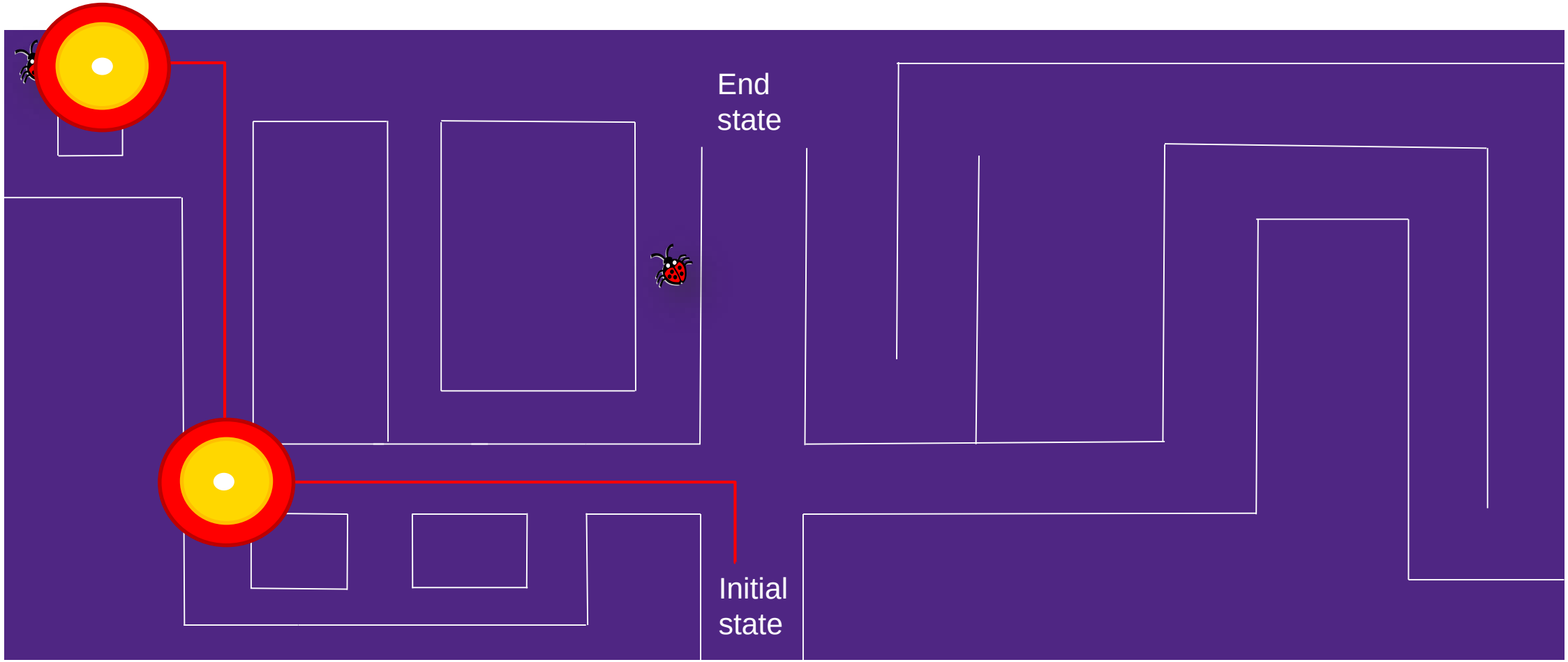
Invariants as Helper Properties – Example (DAC 2014)



Helper Properties – More than Proofs

- Using invariants as helper properties can help prove difficult properties
- Helper properties can also exist in the form of covers
- Deep state space bug hunting utilizes helper covers to guide the search
  - Simulate to interesting state in cover and run formal from there

Deep State Space Bug Hunting



What Makes a Good Helper Property?

- Invariants for proofs
  - Describing relation between signals in design and formal testbench
  - Proving simpler properties on the relationship inside COI of target property
  - Properties related to the main property (for all DAC invariants antecedents were a subset of expression in main property)

- Covers for bug hunting
  - Hit deep extremes in the design (counters full, credit empty etc.)
  - Cover interesting corner cases (long gaps between input/output toggling etc.)

Induction

- Induction is another powerful technique which can help with difficult properties

- An inductive property says that if something is true now, it must be true in the future

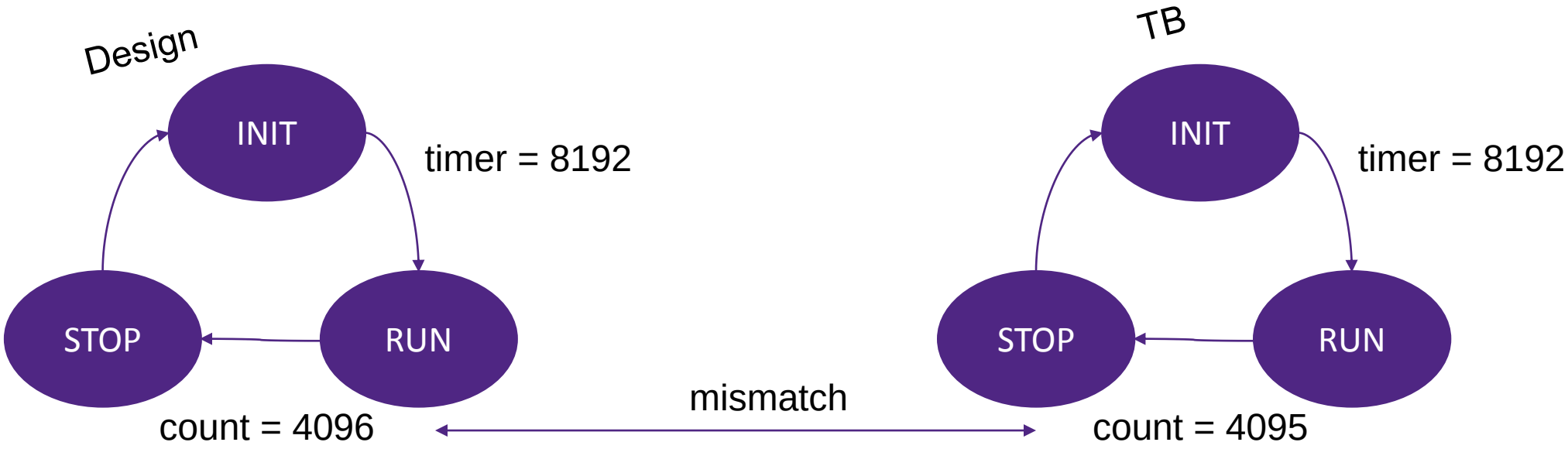
$$a==b \mid\Rightarrow a==b$$

Induction and Initial State Abstraction

- Combining induction with initial state abstraction enhances its power
  - Antecedent relies on consequent currently being true
  - Stops many spurious failures
- Sequential depth problem hugely reduced
- Counterexamples can be reached very quickly

Induction and Initial State Abstraction

Example – Small State Machine

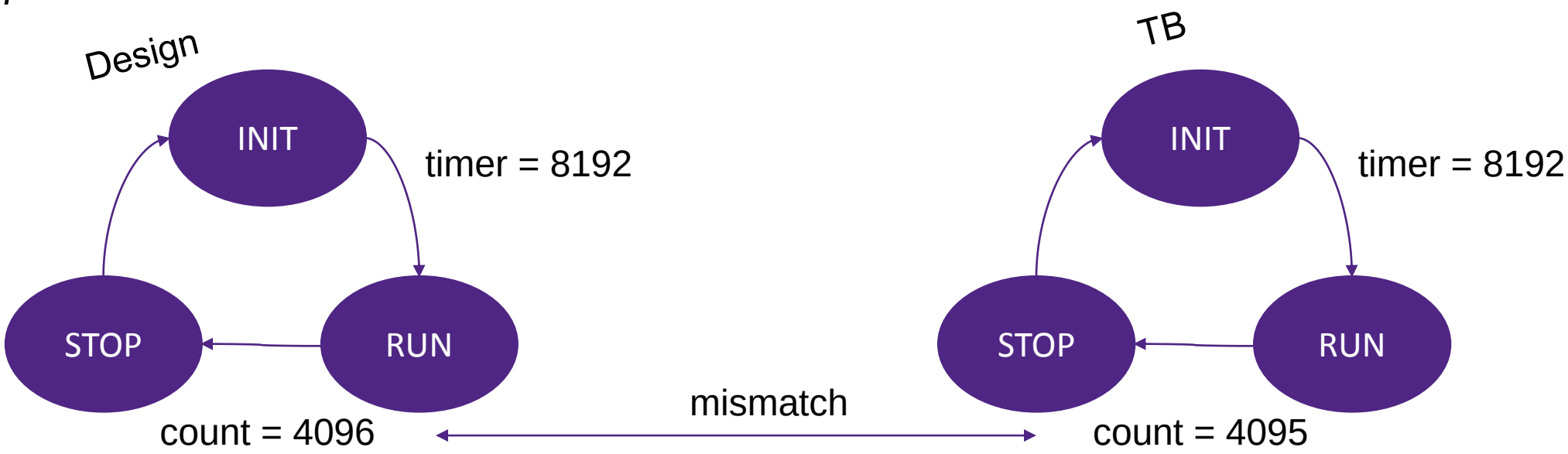


as_state_equal: assert property (design_state == tb_state);

Lots of sequential depth to bug

Induction and Initial State Abstraction

Example – Small State Machine



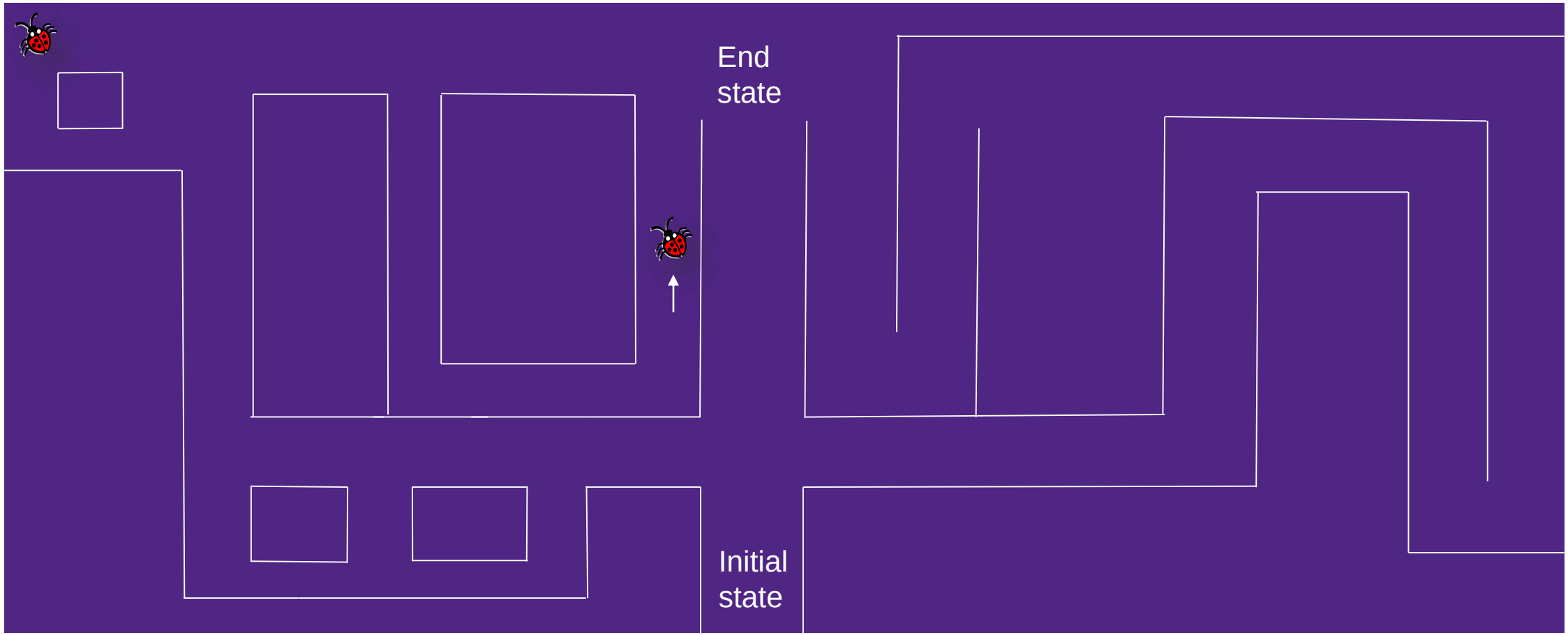
as_ind_state_equal: assert property (design_state == tb_state |=> design_state == tb_state);

No reset to design

Depth 1 constraints that design and tb counter and timer are equal in initial state

Instant 2 cycle CEX

State Space Exploration



Induction and Invariants – Combining the Power

- Finding invariants that will help with convergence is a challenge
- Inductive invariants can be very powerful tools for improving convergence
- Tool capabilities can be used to help with this
  - Find a CEX from a non-reset state
  - Construct an inductive invariant property to prove this CEX cannot happen
  - Add this CEX as helper property and step forward for new CEX

Summary

- Formal is being used on bigger and more complex designs
- For successful formal application on such designs advanced techniques are required
- Induction and invariants are two powerful formal techniques which can enhance convergence
- Finding helpful invariants can be challenging
- Advanced tool features can be used to help develop inductive invariants from non-reset design states
- When all else fails – deep state space bug hunting can find difficult corner cases using formal

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q&A

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

• Introduction	: Sean Safarpour	(20 min)
• Induction & Invariants – Steps to Convergence	: Iain Singleton	(45 min)
• Efficient Formal Modeling Techniques	: Shaun Feng	(45 min)
Break		(10 min)
• Architectural Formal Verification for Coherency	: Syed Suhaib	(45 min)
• Formal Sign-off	: Mandar Munishwar	(45 min)

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Efficient Formal Modeling Techniques

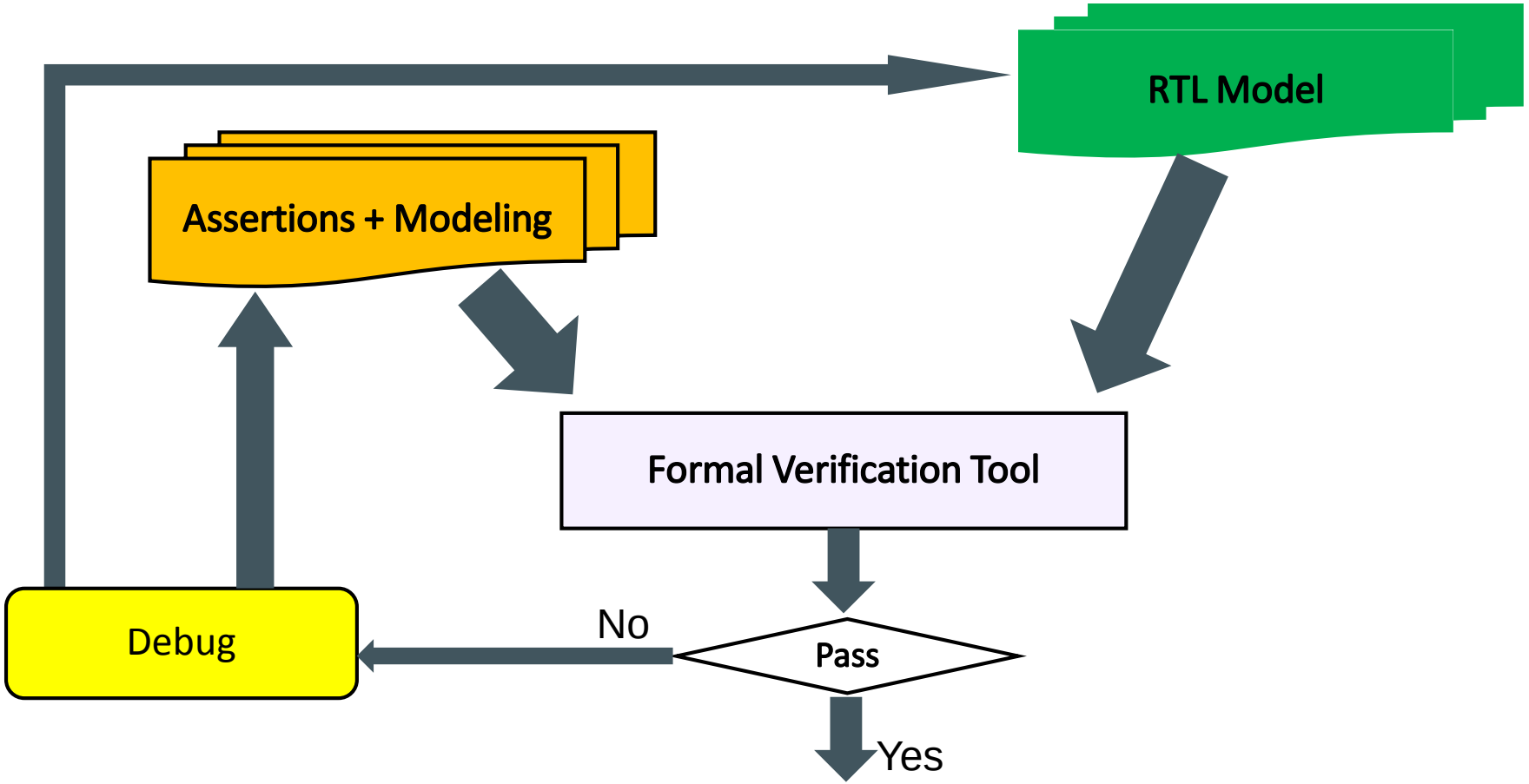
Xiushan “Shaun” Feng
Samsung Austin R&D Center
Email: s.feng@samsung.com

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Formal verification basics
- Abstractions
- Symbolic constants with examples
- Conclusion

Formal Verification Basics



Formal Modeling Goals

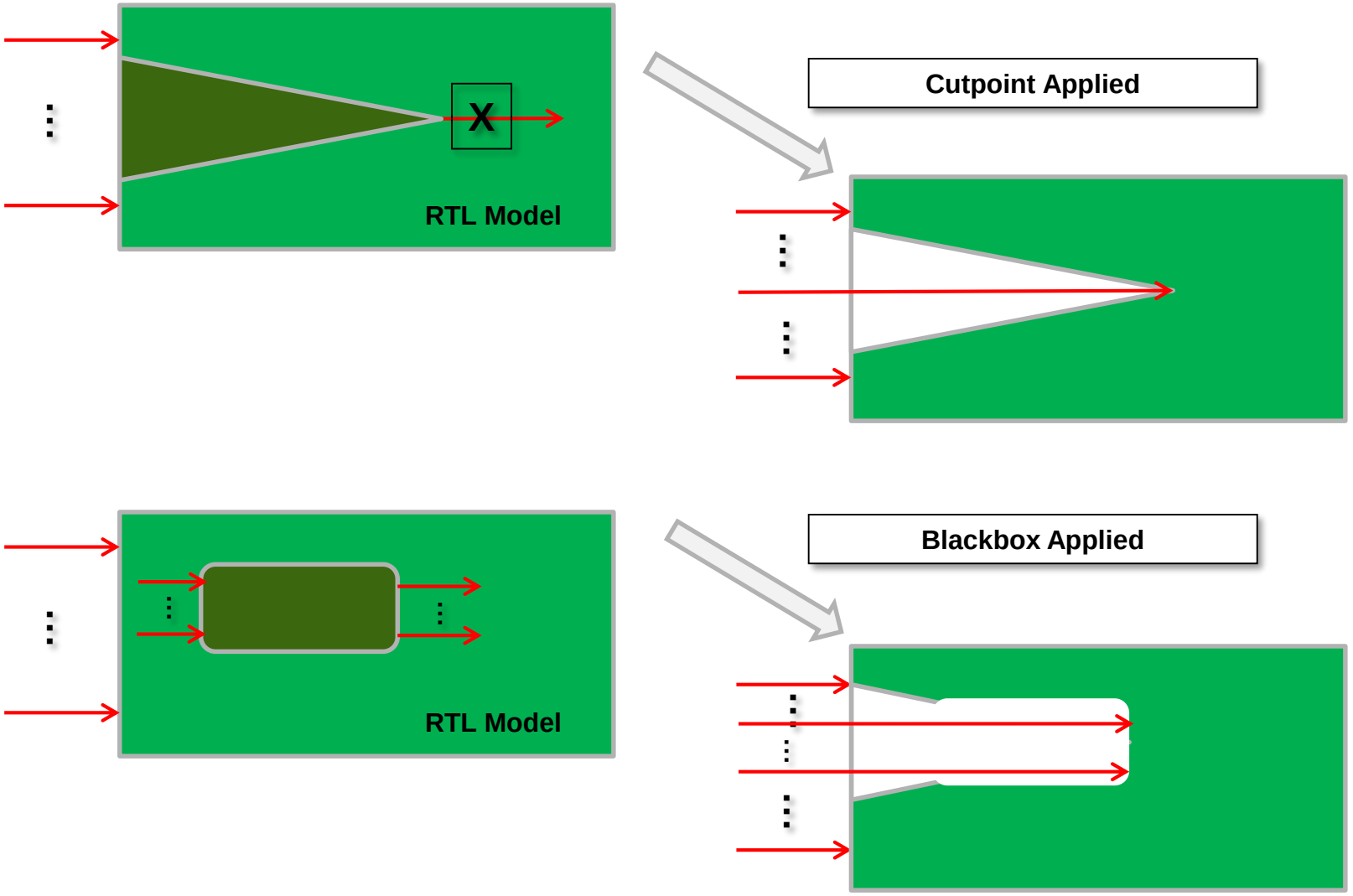
- Goals:
  - Reduce state space – abstraction
  - Cut down the number of assertions
  - Allow formal to quickly find bugs if there is any
- Approaches:
  - Cutpoints/blackboxes/shrinking
  - Assume-guarantee (or divide-conquer)
  - Symbolic constants
  - etc.

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Formal verification basics
- Abstractions
- Symbolic constants with examples
- Conclusion

Cutpoints and Blackboxes



- Can apply to
  - Counters
  - RAMs/ROMs
  - Large arrays
  - Math functions
  - Unnecessary logic
- Conservative
  - No false proven
  - Deep proof bounds
- Side effect
  - False failings
  - May need constraints

Shrunk Design

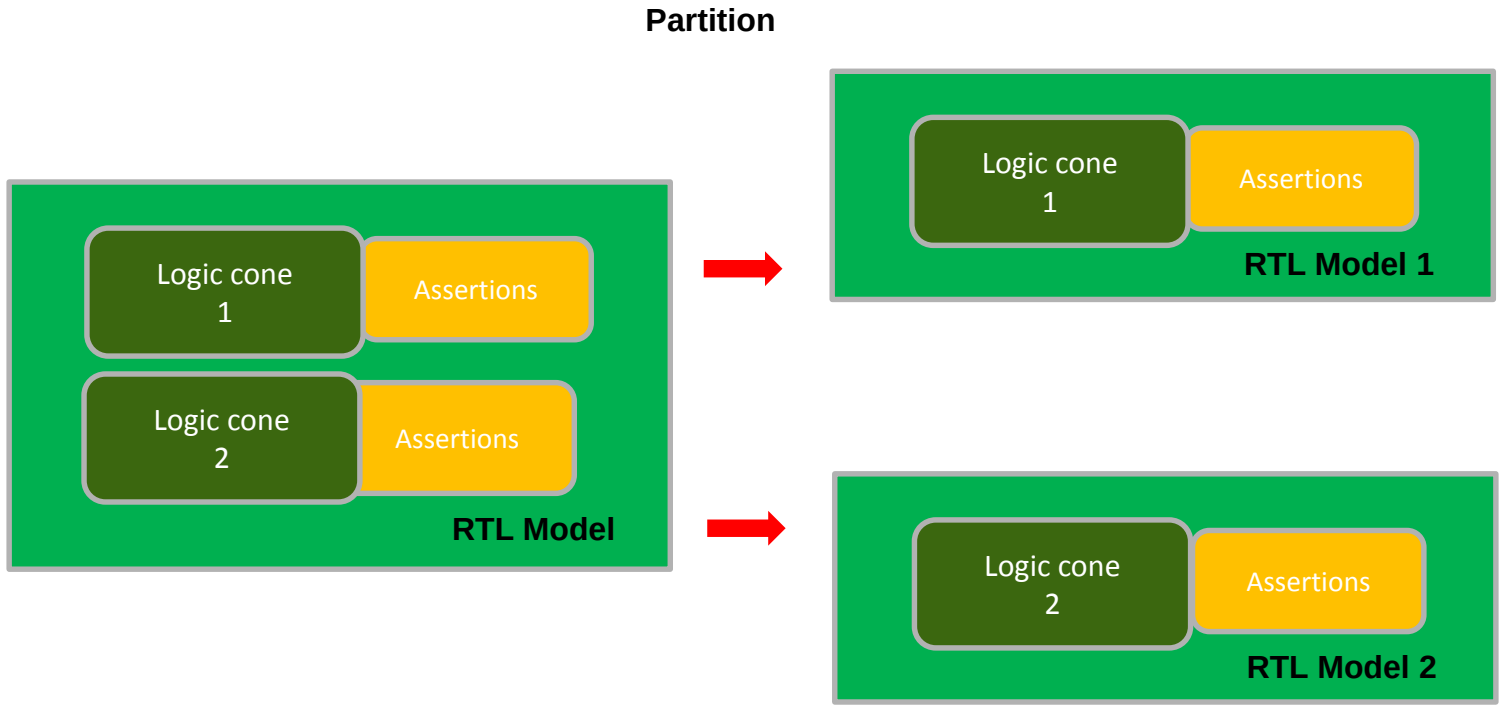
```
module FOO #(parameter bit_iwdth = 10) (
   output reg[bit_width-1:0]  AllocPrt,
   output reg[127:0]          DeAllocData,
   input                      Alloc,
   input [127:0]              Data,
   input                      DeAlloc,
   input                      DeAllocPtr,
…
)
local param addr_size = 2^bit_width
reg [127:0] MEM[addr_size];
…
assign DeAllocData = MEM[DeAllocPtr];
….
endmodule
```

```
module FOO #(parameter bit_width = 1) (
   output reg[bit_width-1:0]  AllocPrt,
   output reg[127:0]          DeAllocData,
   input                      Alloc,
   input [127:0]              Data,
   input                      DeAlloc,
   input                      DeAllocPtr,
…
)
local param addr_size = 2^bit_width
reg  MEM[addr_size];
…
assign DeAllocData = {127{1'b0},MEM[DeAllocPtr];
….
endmodule
```

- Address space
  - Cache coherence needs only one address
- Data size
  - 1 bit may be enough for data integrity check
- FIFO
  - Depth of FIFO can be reduced
  - IO flopped delay can be removed
- Other symmetric structures

Assertion/Design Partition

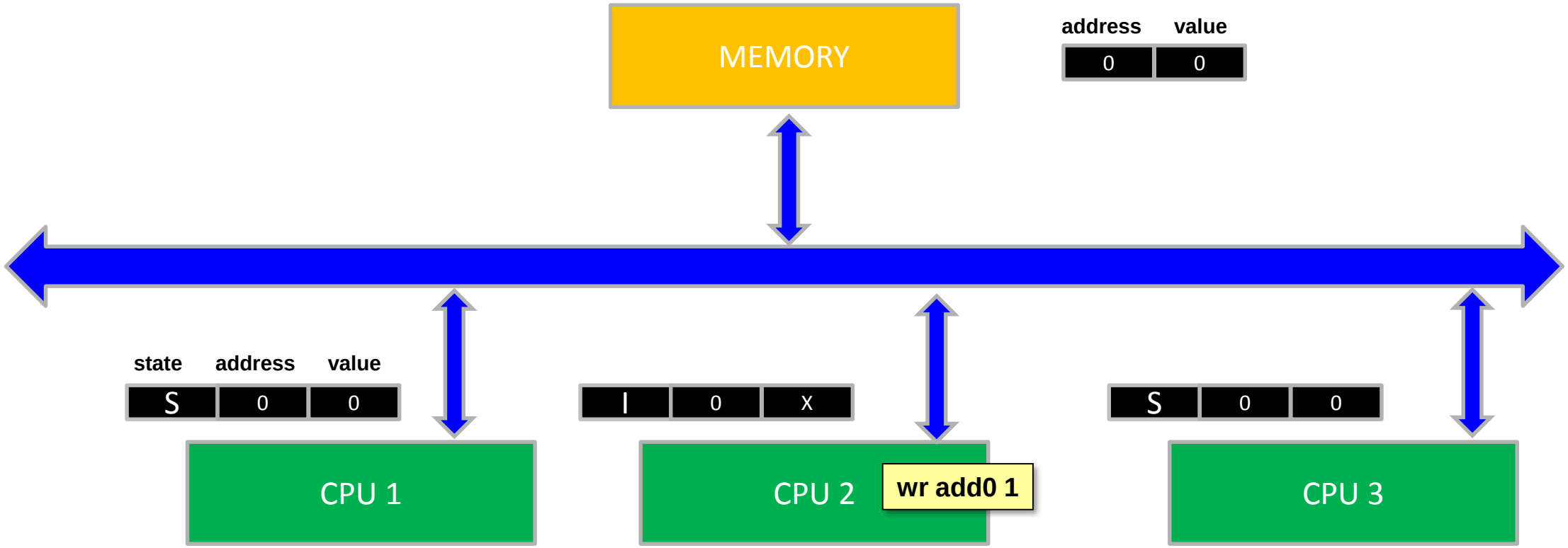
- Assertion partition
  - Grouping assertions with same COI
  - Using proven assertions as assumptions
- Design partition
  - Using assertion groups to partition design
  - One formal test for each partition



Preloading

- Instead of starting formal at initial state, we can start at a valid pre-defined state
  - Configuration registers
  - Counters
  - FSM
  - Cache/memory
  - A witness trace of a cover property

Preloading MESI State



Counter Abstraction

- Not all values of a counter are valid.
  - 32bit counter has 2^32 possible values
  - Abstract away the counter and assume possible values.
- Initial values of counters
  - Usually, counters are initialized to predefined values (e.g, 0)
  - Counter-example can happen with a large counter value – a long trace to hit
  - Counter initial value abstraction helps to shorten the trace

Counter Abstraction Example

Initial Value Abstraction

```
reg [bit_width-1:0]   counter;

always_ff @(posedge clock) begin
  if (reset)
   `ifdef FORMAL_ON
   `else
       counter <= 'b0;
    `endif
  else if (…)
    counter <= counter + 1'b1;
end
```

Counter Value Abstraction

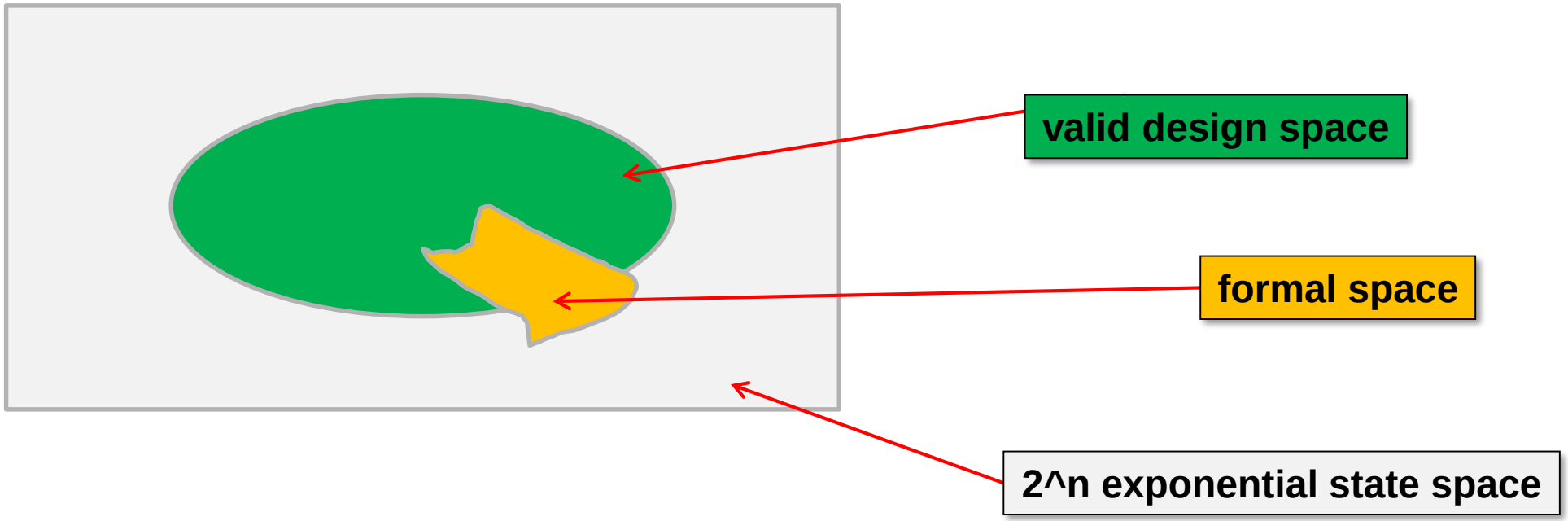
TCL control file:
cutpoint DUT.counter
assume {condition |-> DUT.counter inside {**0, 1, 2, 4**}}

Assume-Guarantee

- General approach:
  - Break down a big problem into a few sub-problems
  - Assume sub-problems
  - Prove big problem with added assumptions
  - Prove sub-problems
- Techniques can be used:
  - Design partition
  - Blackboxes
  - Cutpoints
  - Assertion re-writing

Over Constraints Used as Abstraction

- Over constraints are not always bad
  - Smaller state space
  - Finer-grain control of inputs
- formal test bench can have both over and under constraints



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

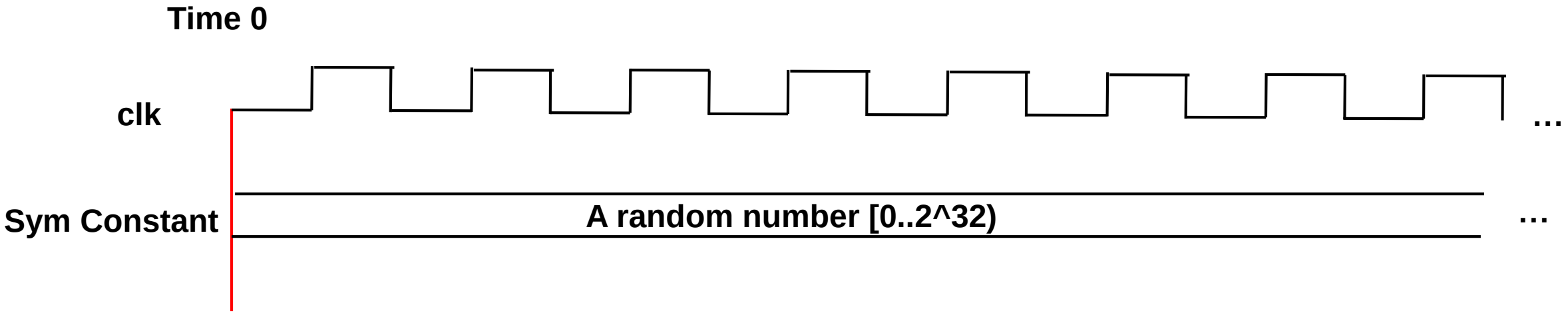
Agenda

- Formal verification basics
- Abstractions
- Symbolic constants
- Conclusion

Symbolic Constants

- A random value after reset
- Will hold its value throughout the whole formal proof

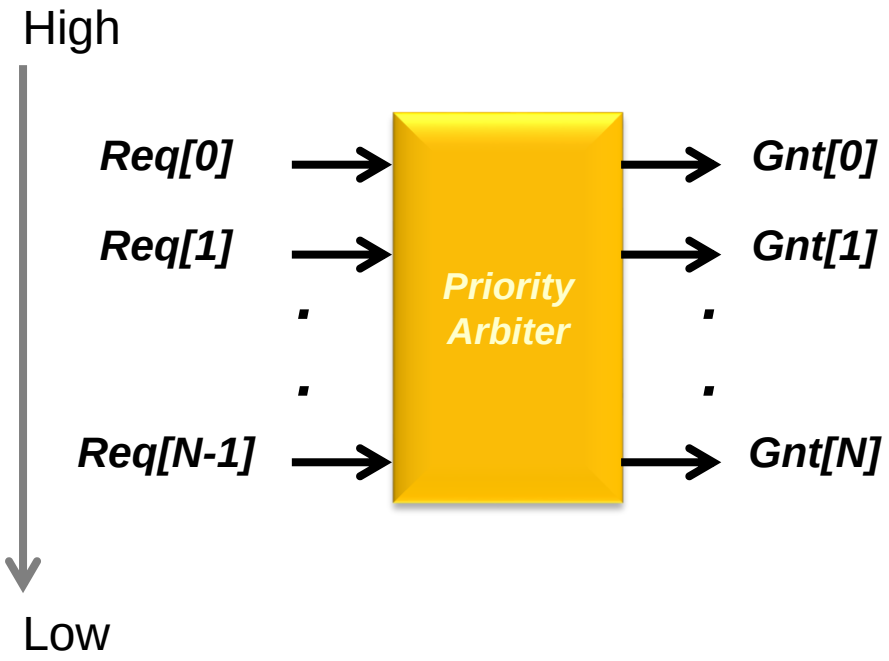
```
(@ posedge clk) ##1 $stable(SymC[31:0])
```



Symbolic Constant Examples

- Priority Arbiter
- Round Robin Arbiter
- In order transport example

Priority Arbiter




```
// if m<n, Req[m] has higher prority than Req[n]
// if there is a Req[m], Req[n] cannot be granted
// without  grant m first
property priority_pair (m,n);
  @(posedge clk) disable iff (~reset_n)
     not ( ((m < n) && req[m] & !gnt[m])
          throughout (gnt[n])[->1]));
endproperty


generate
 for (genvar m = 0; m<=N; m++) begin
  for (genvar n = 0; n<=N; n++) begin
   assert property (priority_pair(m,n));
  end
 end
endgenerate
```

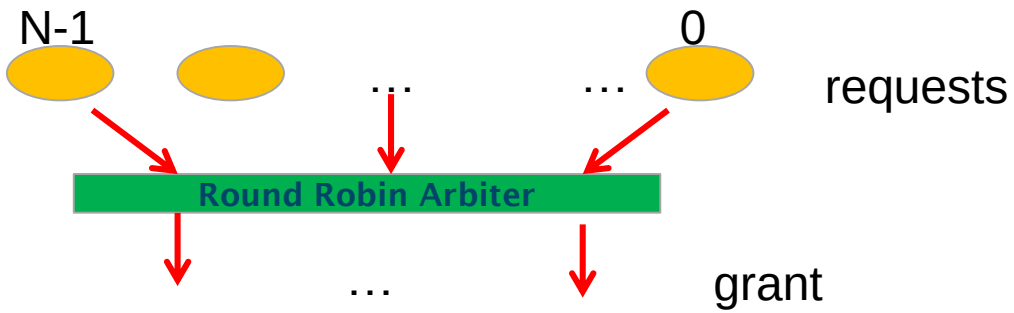
Use Symbolic Constants

```
localparam      WIDTH   = $clog2(N);
logic   [WIDTH-1:0]          m, n;
ASM_SYM_CONST_m_n: assume property (@(posedge n_clock) disable iff
(!n_resetb)
    ##1 $stable(m) && $stable(n) && m < N && n < N);

AST_PRI_ARB: assert property (@(posedge n_clock) disable iff (!n_resetb)
    not (strong(((m < n) && req[m] & !gnt[m]) throughout (gnt[n])[-
>1])));
);
```

- M, N are random values at reset, but will hold the values after reset.

Round Robin Arbiter

- Strong fairness
- Severed request gets the lowest priority
- Rotated priority



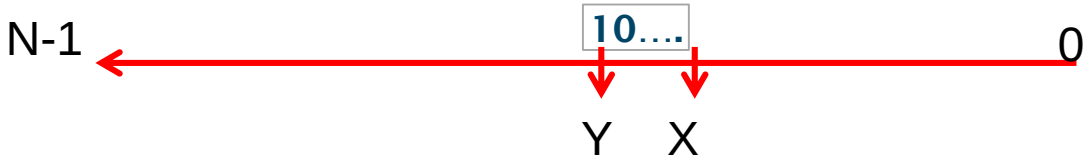
Assertion Checks

- Fairness
- One hotness
- Round robin (rotated priority)

Cases

Req X is just served, expect to serve Y later

Case 1



Y > X, i ∈(X, Y), req[i] ==0

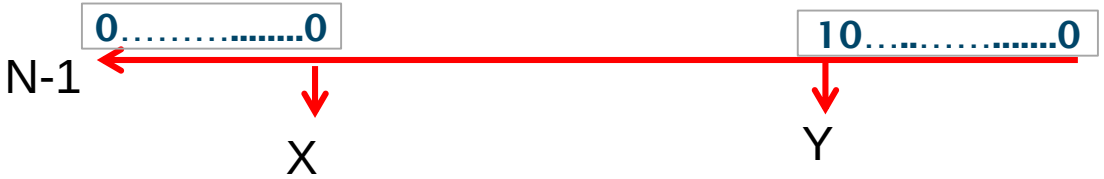
Case 2

X > Y, Y==0,
i ∈(X, N-1], req[i] ==0

Case 3

X > Y, X==N-1,
i ∈[0, Y), req[i] ==0

Case 4



N>X > Y>0,
i ∈(X, N-1] ∪[0, Y), req[i] ==0

Assertions

```
localparam      WIDTH   = $clog2(N);
logic   [WIDTH-1:0]         X, Y;
ASM_SYM_CONST_X_Y: assume property (@(posedge n_clock) disable iff (!n_resetb)
    ##1 $stable(X) && $stable(Y) && X < N && Y < N);
generate
   for (genvar i = 0; i < N; i++) begin : location_asm
     ASM_CASE1: assume property (@(posedge n_clock) disable iff (!n_resetb)
        Y > X && Y>i && i>X |-> Req[i]==0);
     ASM_CASE2: assume property (@(posedge n_clock) disable iff (!n_resetb)
        X > Y && Y ==0 && i > X |-> Req[i]==0);
     ASM_CASE3: assume property (@(posedge n_clock) disable iff (!n_resetb)
        X > Y && X==N-1 && i < Y |-> Req[i]==0);
     ASM_CASE4: assume property (@(posedge n_clock) disable iff (!n_resetb)
        X > Y && (i > X | i < Y) |-> Req[i]==0);
   end
endgenerate

AST_RR_ARB: assert property (@(posedge n_clock) disable iff (!n_resetb)
    ##1 $past(Req[X] && Gnt[X]) && Req[Y] && Y !=X
   |-> $onehot(Gnt) && Gnt[Y]
);
AST_RR_ONEHOT: assert property (@(posedge n_clock) disable iff (!n_resetb)
    $onehot0(Req)
    |-> Gnt  == Req
);
AST_RR_FAIR: assert property (@(posedge n_clock) disable iff (!n_resetb)
      not((Req[X] &&~Gnt[X])[*N])
);
```

Zoom in Fairness Assertion

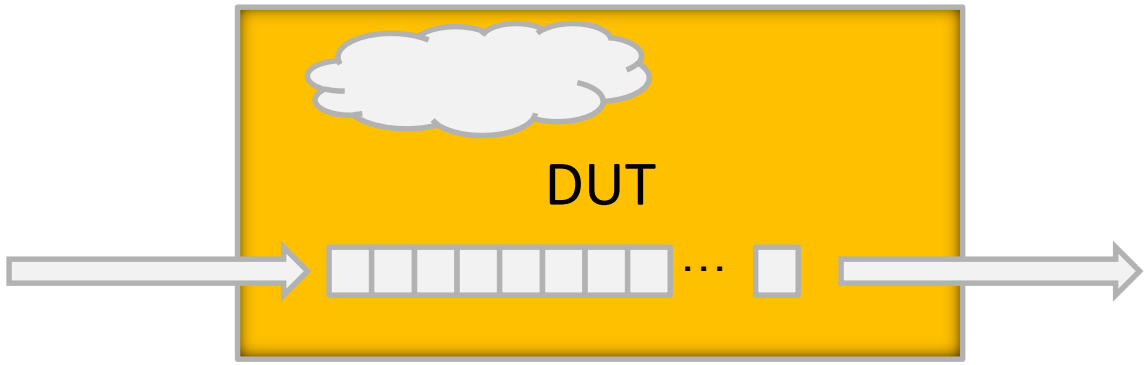
```
AST_RR_FAIR: assert property (@(posedge n_clock) disable iff
(!n_resetb)
      not((Req[X]&& ~Gnt[X])[*N]));
```

- What happened if N is very big.

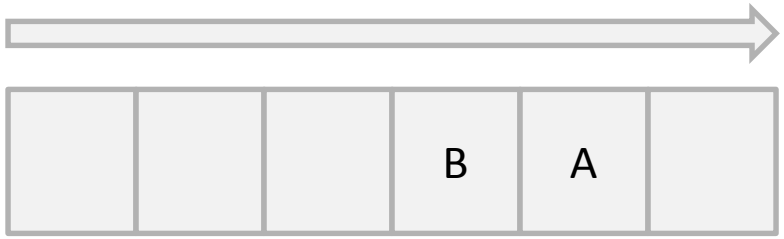
```
AST_RR_FAIR: assert property (@(posedge n_clock) disable iff
(!n_resetb)
     X!=Y |-> not(Req[X] throughout Gnt[Y][->2])
);
```

In Order Transport

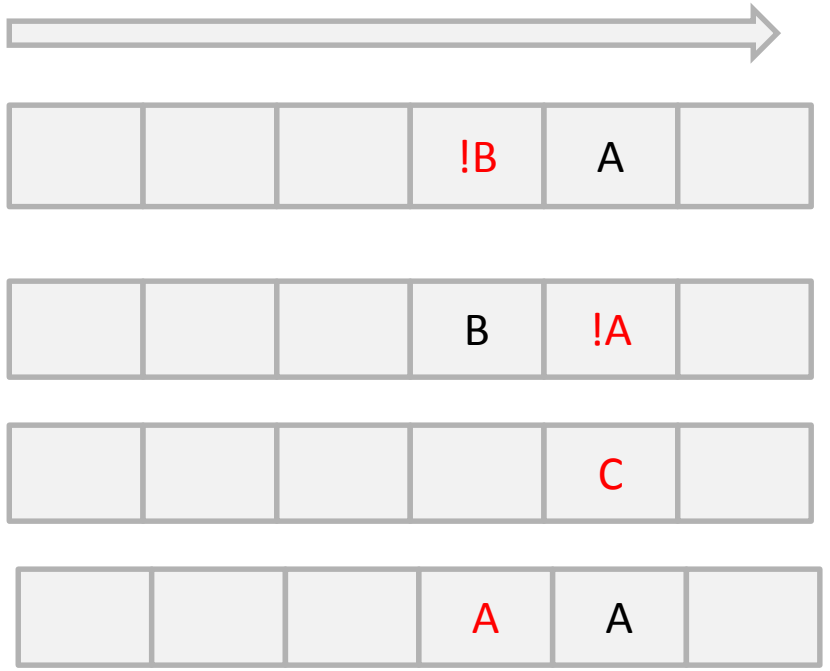
- Data comes out in order
- No drop of data
- No spurious data comes out
- No duplication of data



Inputs



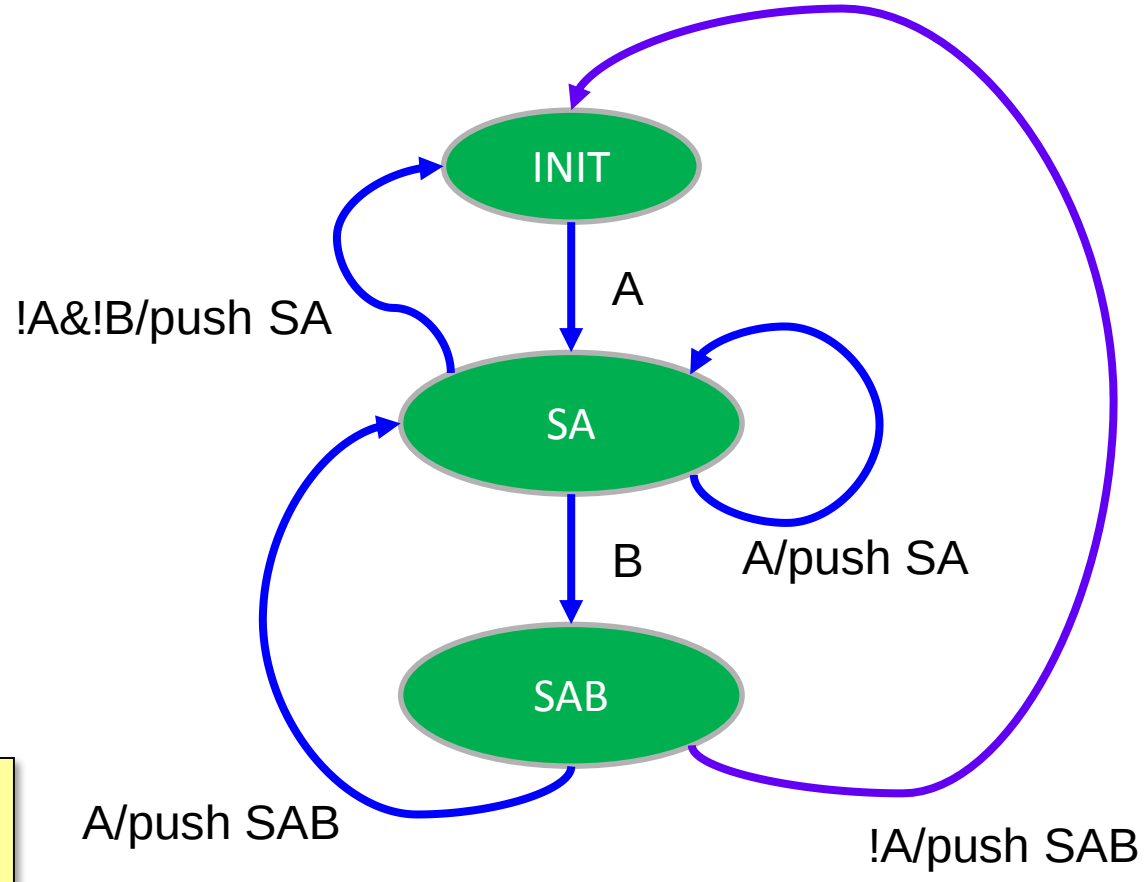
Bad outputs



Modeling

- A standard FIFO is used
  - With full/empty state
- Input/output FSMs
- 3-state FSM
  - SA: seen A
  - SAB: seen A, B
  - INIT: initial state

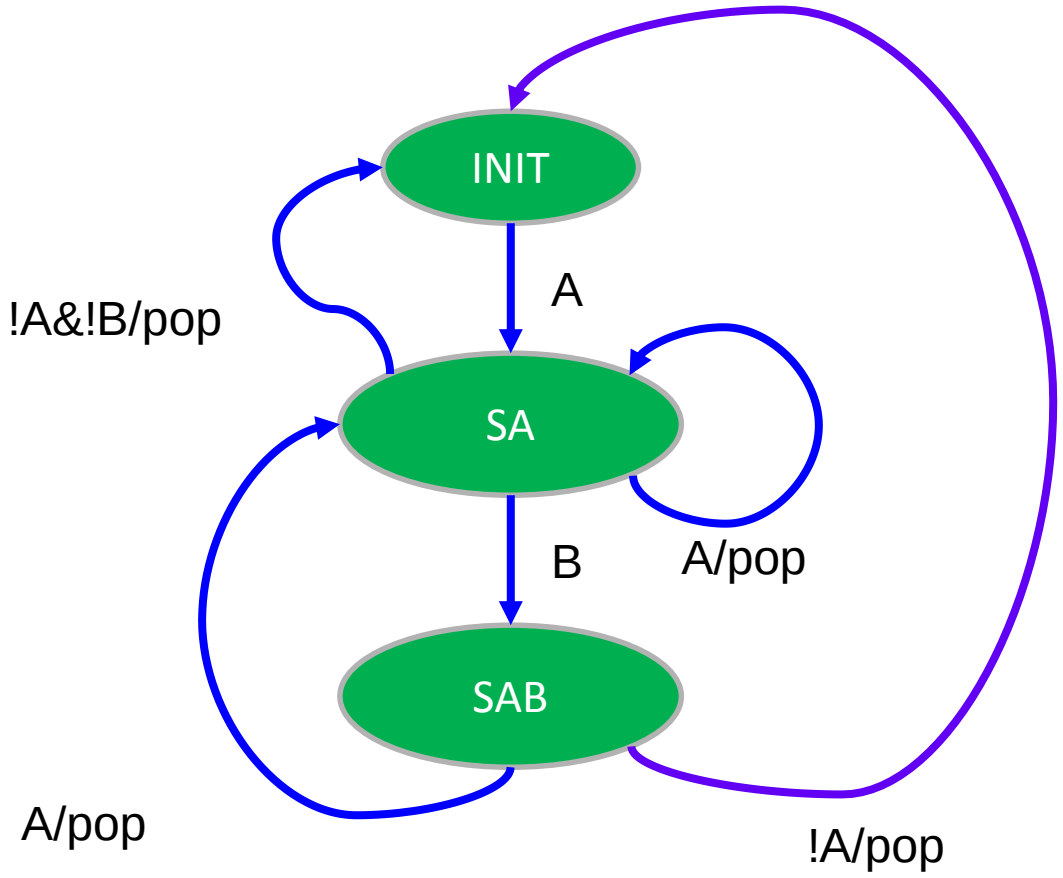
Input monitor state machine




```
assume property (##1 $stable(A) && $stable(B)
                 && A!= B);
```

Modeling – Cont.

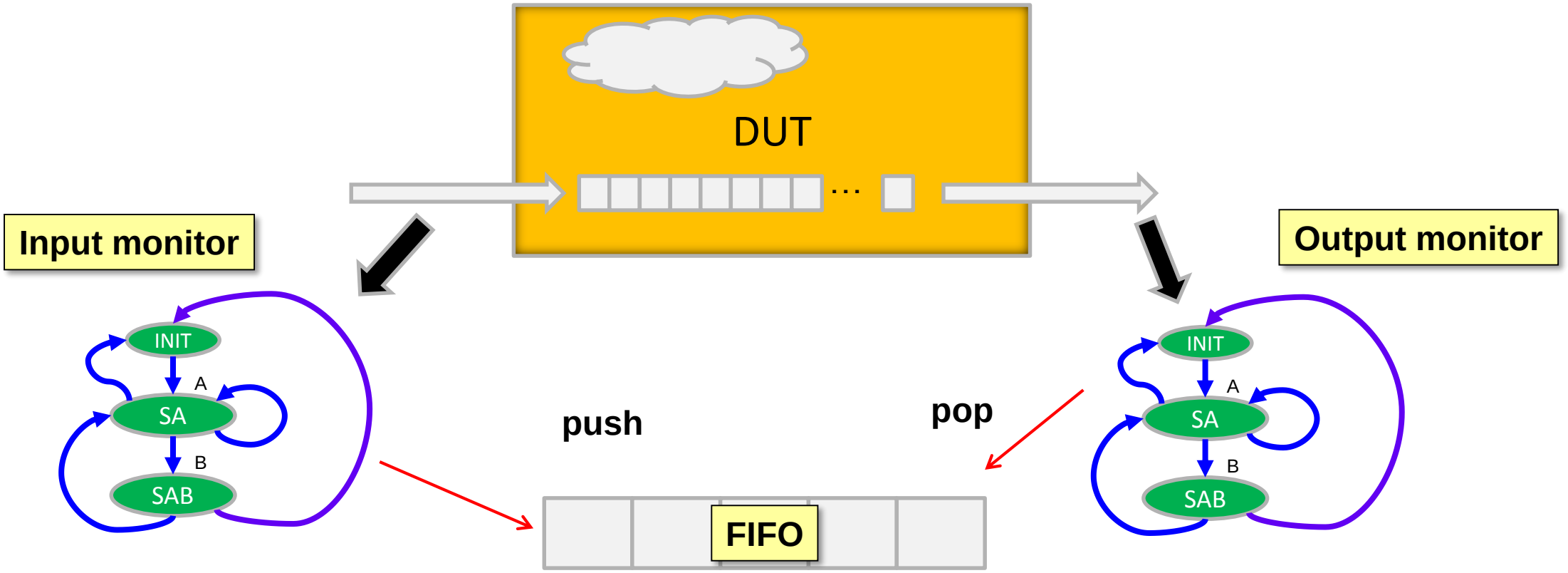
Output monitor state machine



Flow control

```
ASM_EOC_COND: assume property (
    fifo.full || rand_stop
    |->
    in != A && in!= B && in_vld &&
completed
);

ASM_EOC: assume property (
    completed |=> completed && !in_vld
);
```

Implementation



Checker:

```
pop |-> fifo.out == output_monitor.state
```

Symbolic Constant in Simulation

- Symbolic constants can not be used directly for simulation.
  - **$stable**() can be replaced by a random number.

```
localparam      WIDTH   = $clog2(N);
logic   [WIDTH-1:0]         m;
`ifdef FORMAL
ASM_SYM_CONST_m: assume property (@(posedge n_clock) disable iff (!n_resetb)
             ##1 $stable(m) && m< N
);
`else
  initial begin
    assert(std::randomize(m));
  end
`endif
AST_PRI_ARB: assert property (@(posedge n_clock) disable iff (!n_resetb)
    not ((req[m] & !gnt[m])[*N])
);
```

Conclusion

- Efficient formal verification modeling techniques are crucial to formal verification
  - Simplify formal modeling code
  - Improve runtime
- Abstraction is the key
  - Abstractions with cost (false counter examples)
  - Understand designs and find the right balance

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q&A

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

• Introduction	: Sean Safarpour	(20 min)
• Induction & Invariants – Steps to Convergence	: Iain Singleton	(45 min)
• Efficient Formal Modeling Techniques	: Shaun Feng	(45 min)
Break		(10 min)
• Architectural Formal Verification for Coherency	: Syed Suhaib	(45 min)
• Formal Sign-off	: Mandar Munishwar	(45 min)

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Architectural Formal Verification of Coherency Manager

Syed Suhaib - Nvidia Corp.

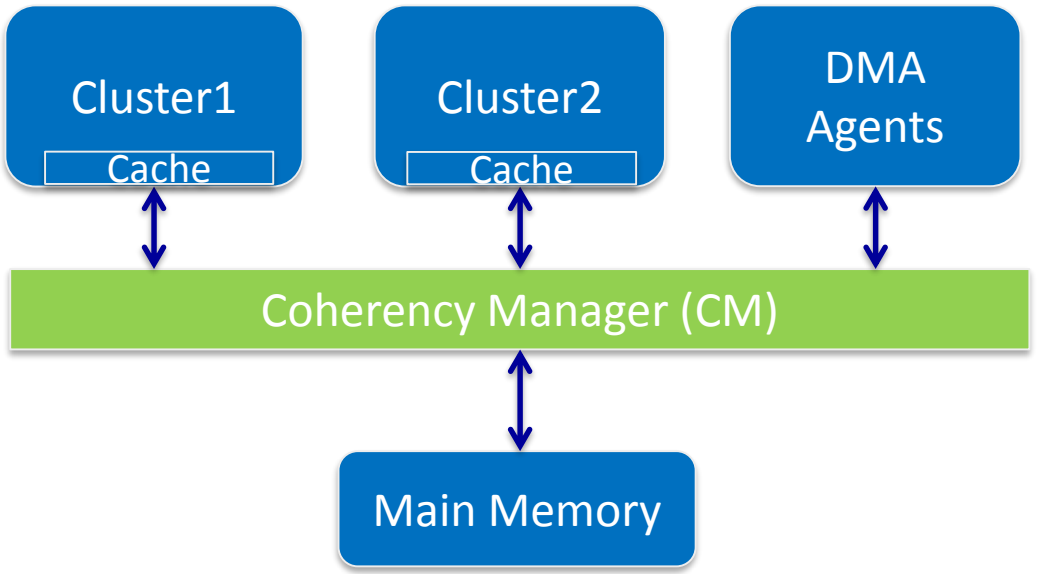
Email: ssuhaib@nvidia.com

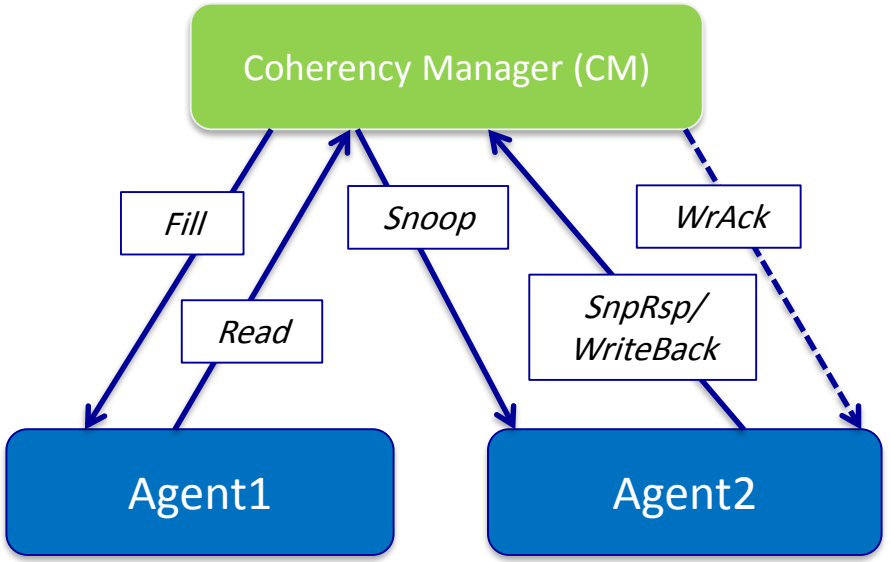
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Coherency Manager
- Verification Methodology
- Coherency Manager’s Architectural Model
- Results

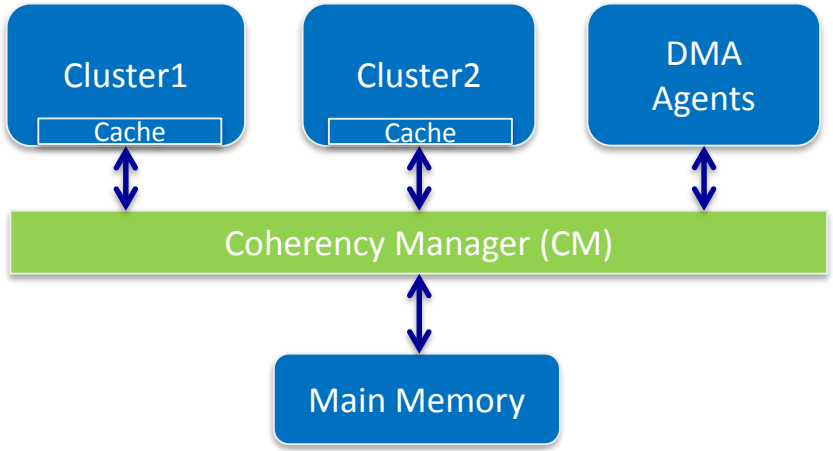
Coherency Manager





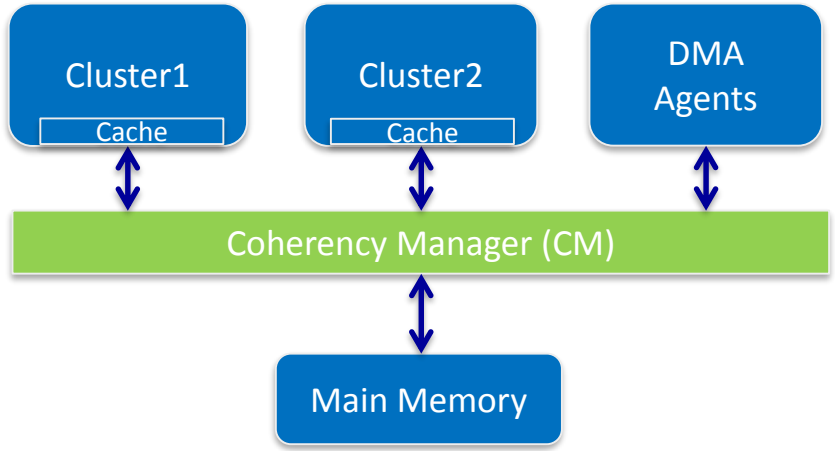
Verification Challenges

- High Complexity
- Large DUT
- Traditional Simulation Doesn't Work Well!
  - Slow
  - Coverage Challenges
  - Stub models for multiple Clusters
    - Tricky



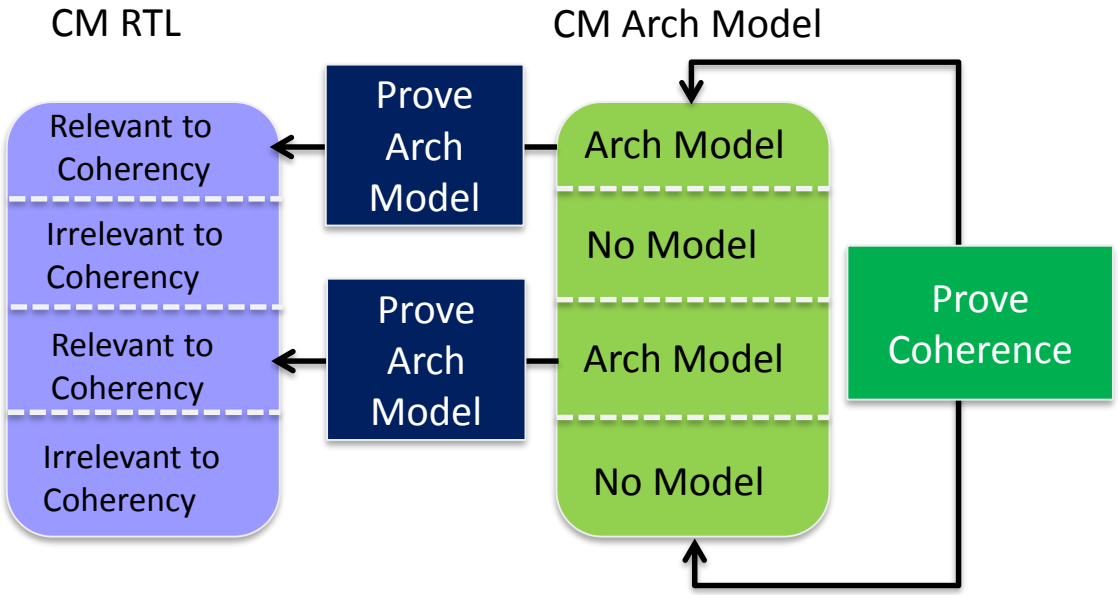
Verification Challenges

- Formal Verification (FV)
  - Impractical to apply FV on entire system
    - State space
  - May create a custom setup
    - Black-box sub-units and add assumptions
    - Onion-peeling effort
      - Getting rid of non-relevant micro-arch details

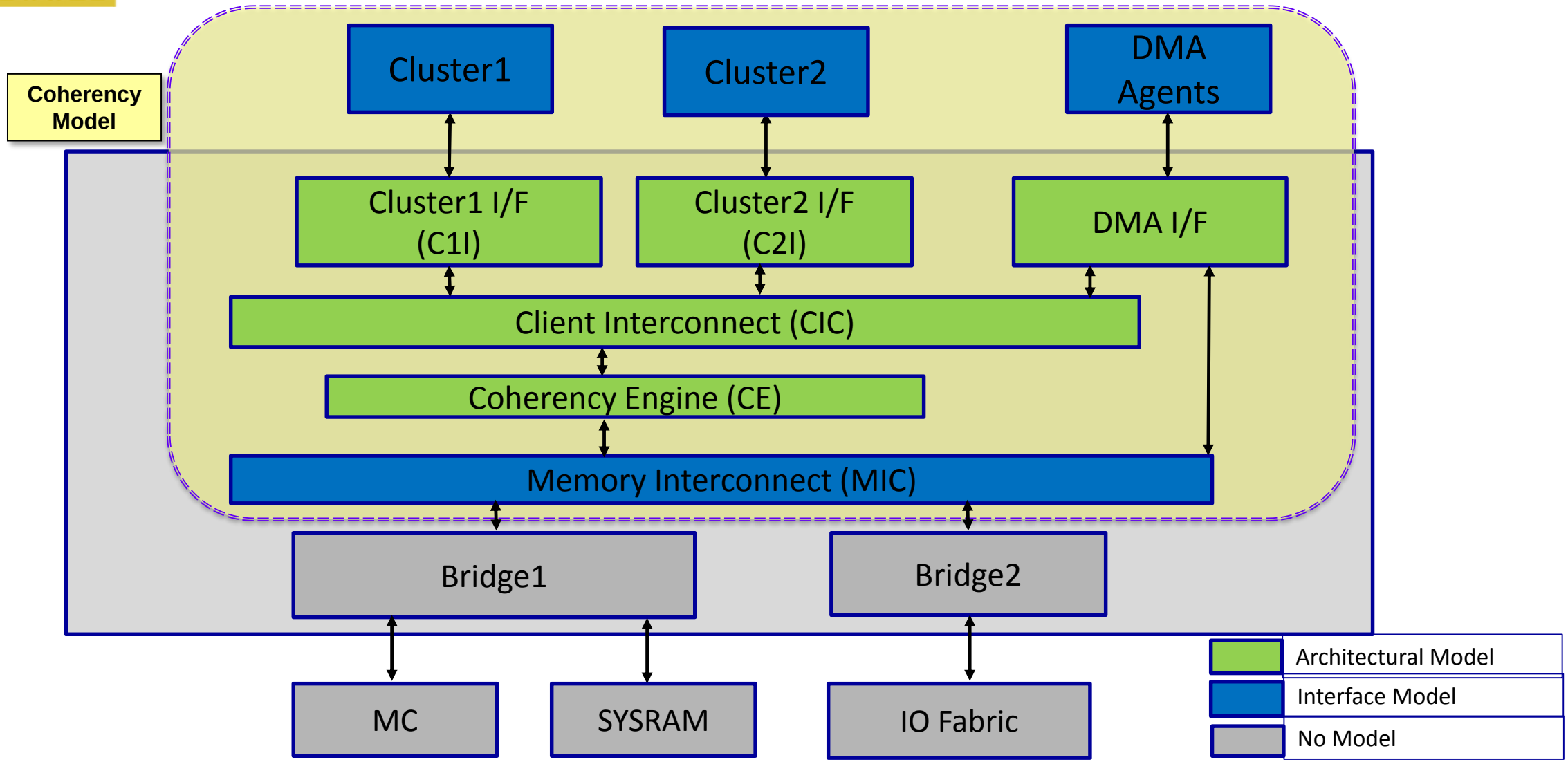


Steps of Architectural Verification

1. Code Architectural models of Coherency Manager components affecting coherency
2. Prove Coherence on interconnection of Architectural models (FPV)
3. Prove Architectural models against Coherency Manager RTL subunits (FPV)

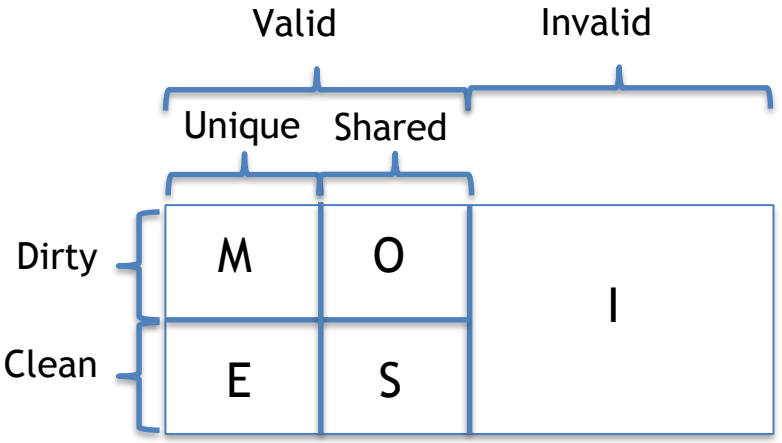


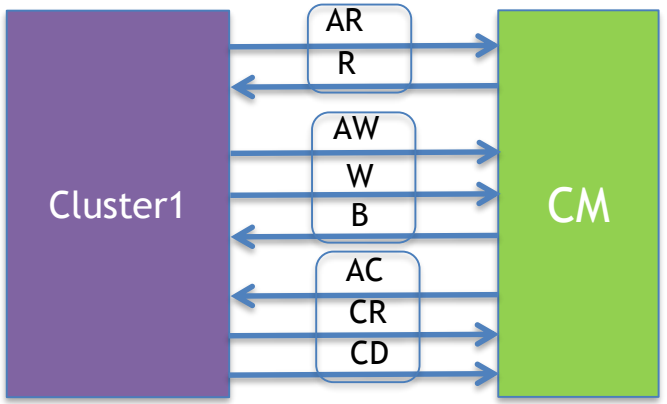
Coherency Manager Block Diagram

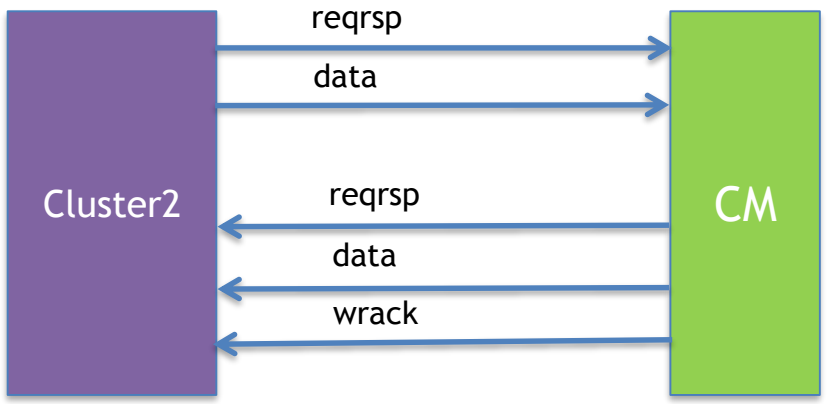


Cluster1 vs. Cluster2 Interface Model

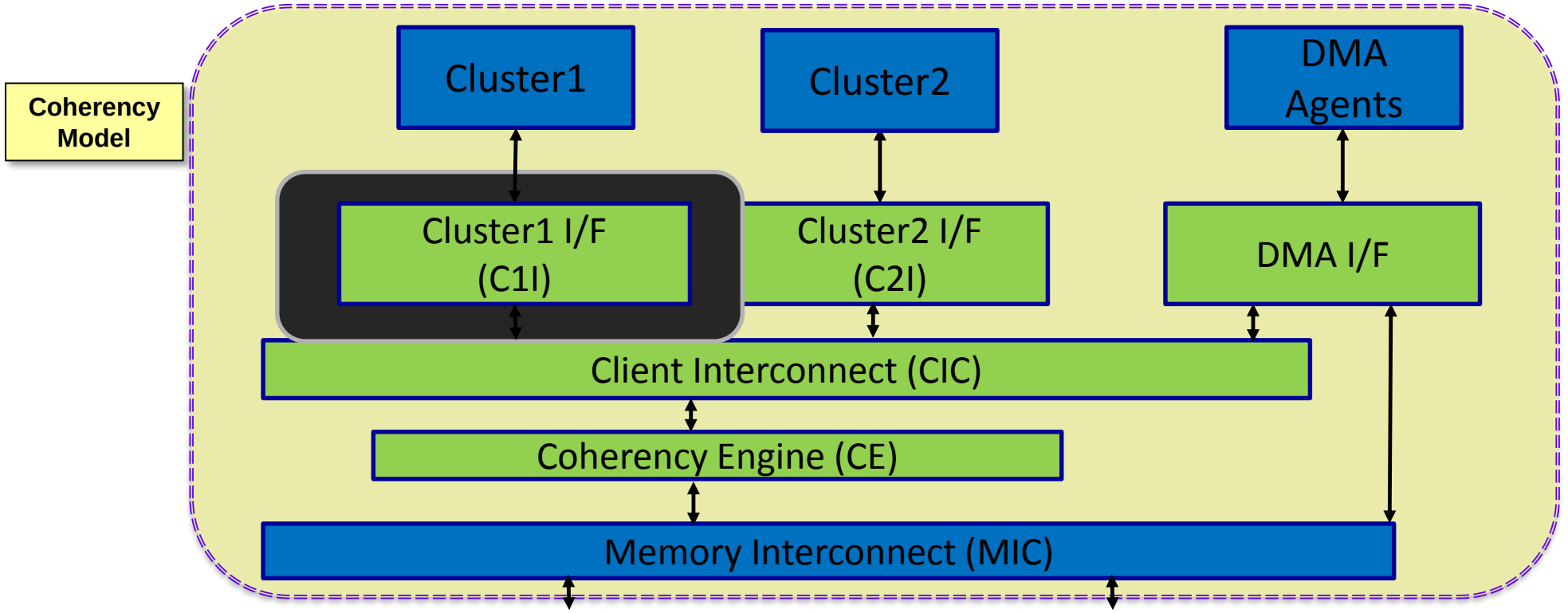
	Cluster1	Cluster2
Interface	ACE	Proprietary
Coherency Protocol	MOESI	MESI
Cache-line Model	Oski ACE VIP	Coded in-house



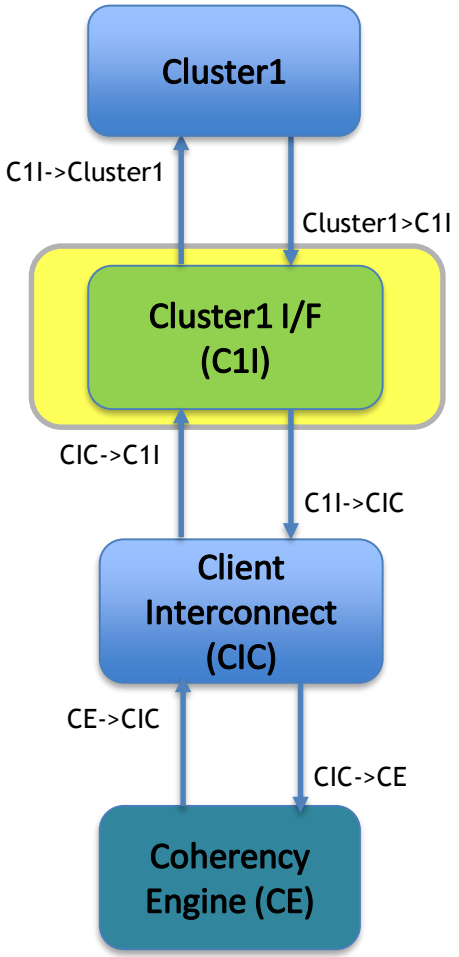




Cluster1 Interface (C1I) Model

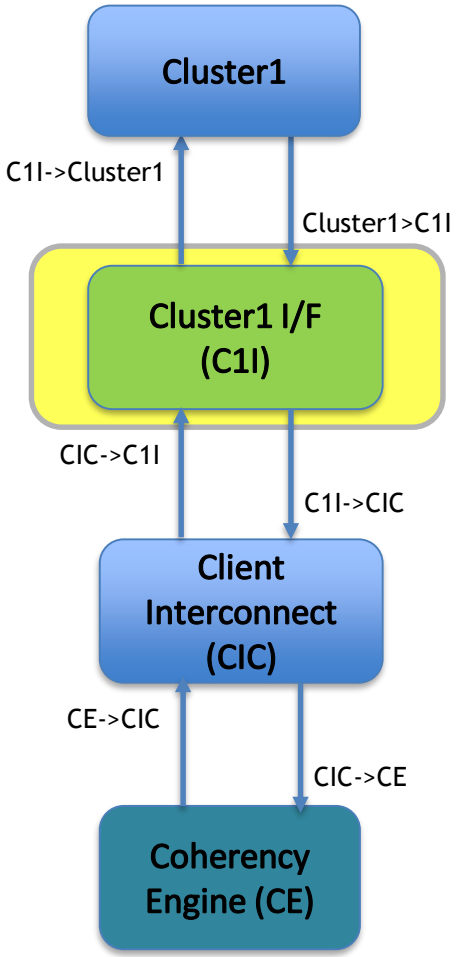


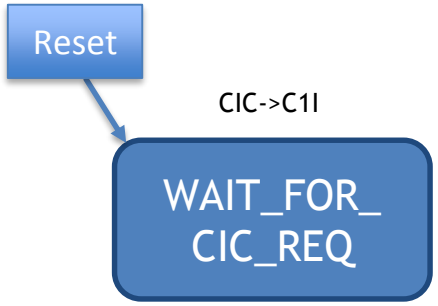
Cluster1 Interface (C1I) Model



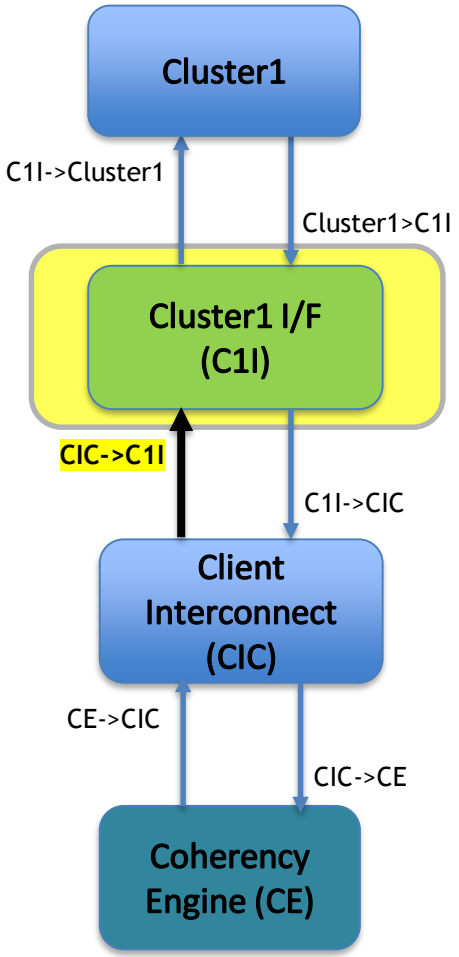
- Tracks progress of requests for a particular cache-line
  - Read Tracker
  - Write Tracker
  - Snoop Tracker
- Trackers can be replicated for multiple cache-lines

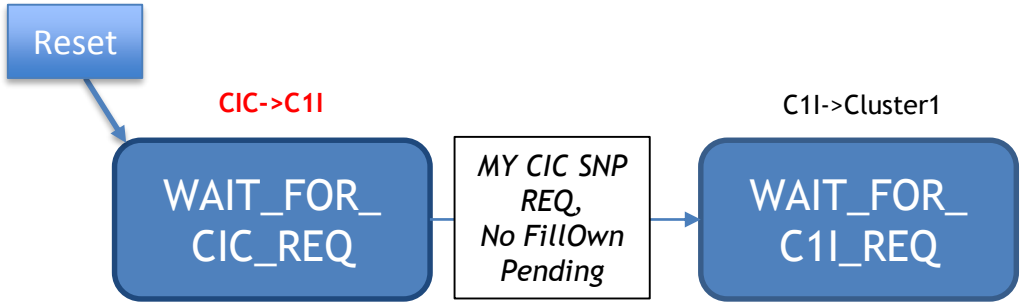
Snoop Tracker



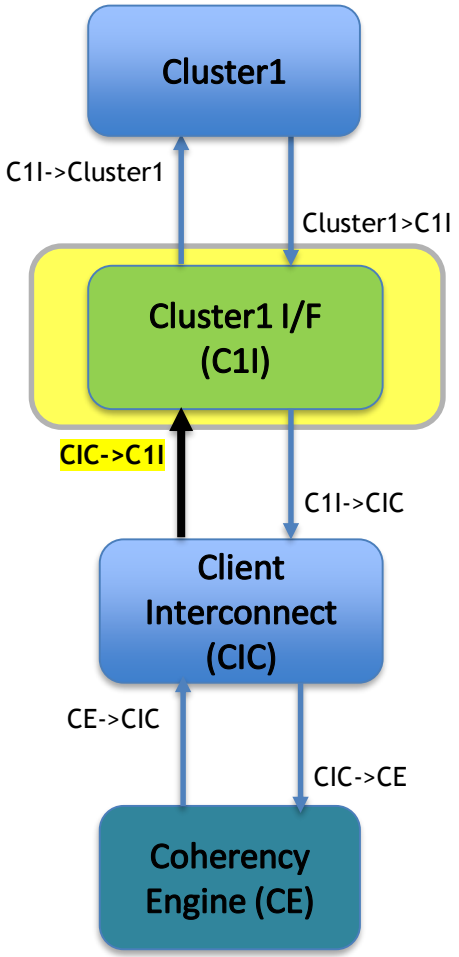


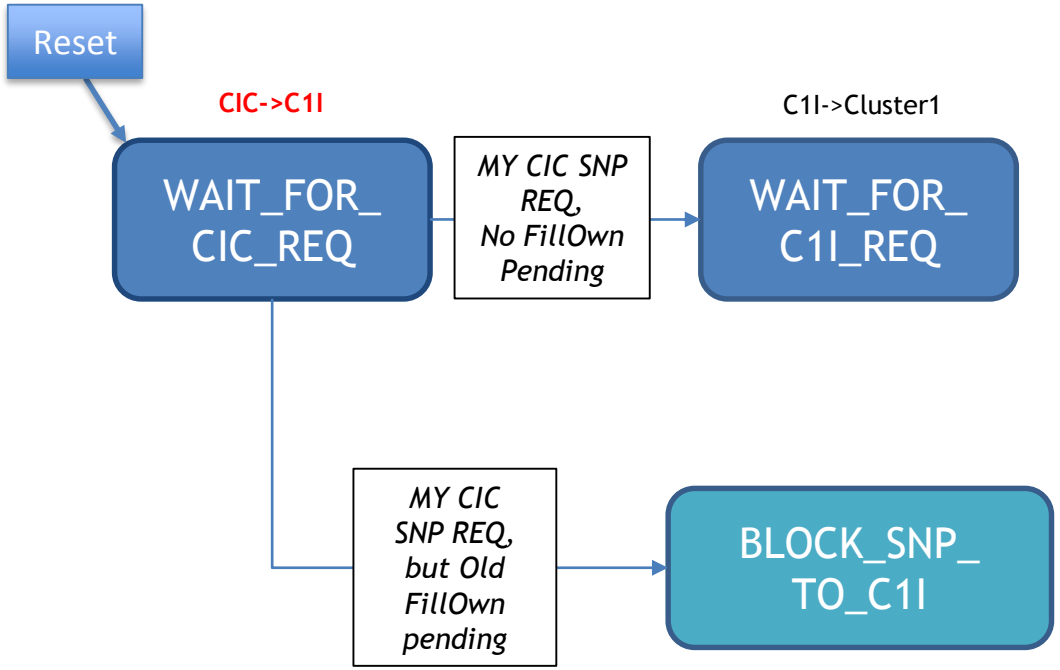
Snoop Tracker



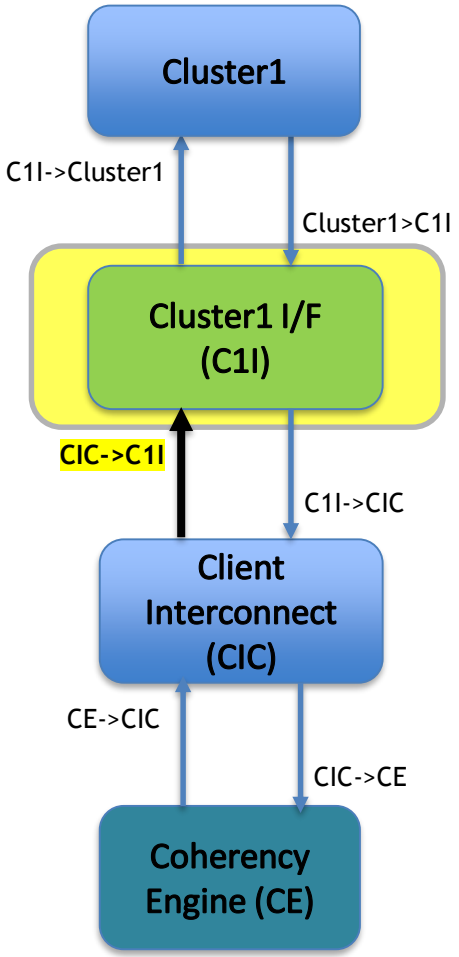


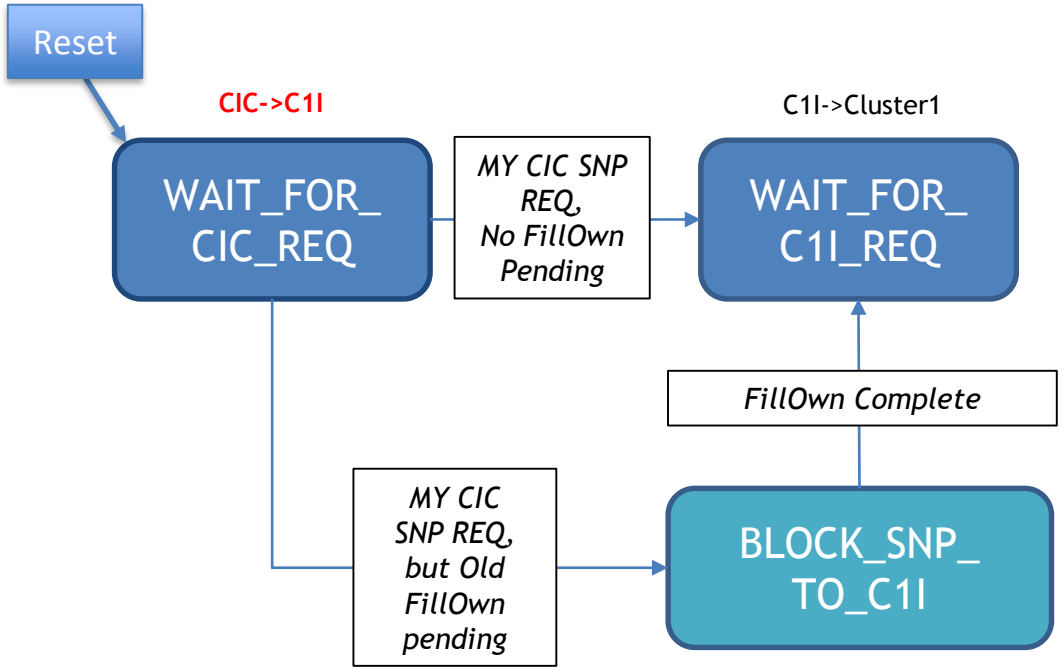
Snoop Tracker



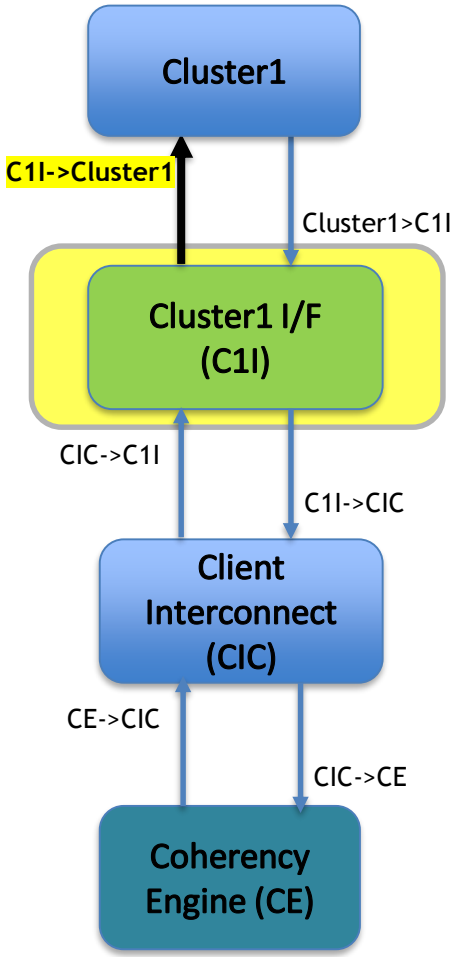


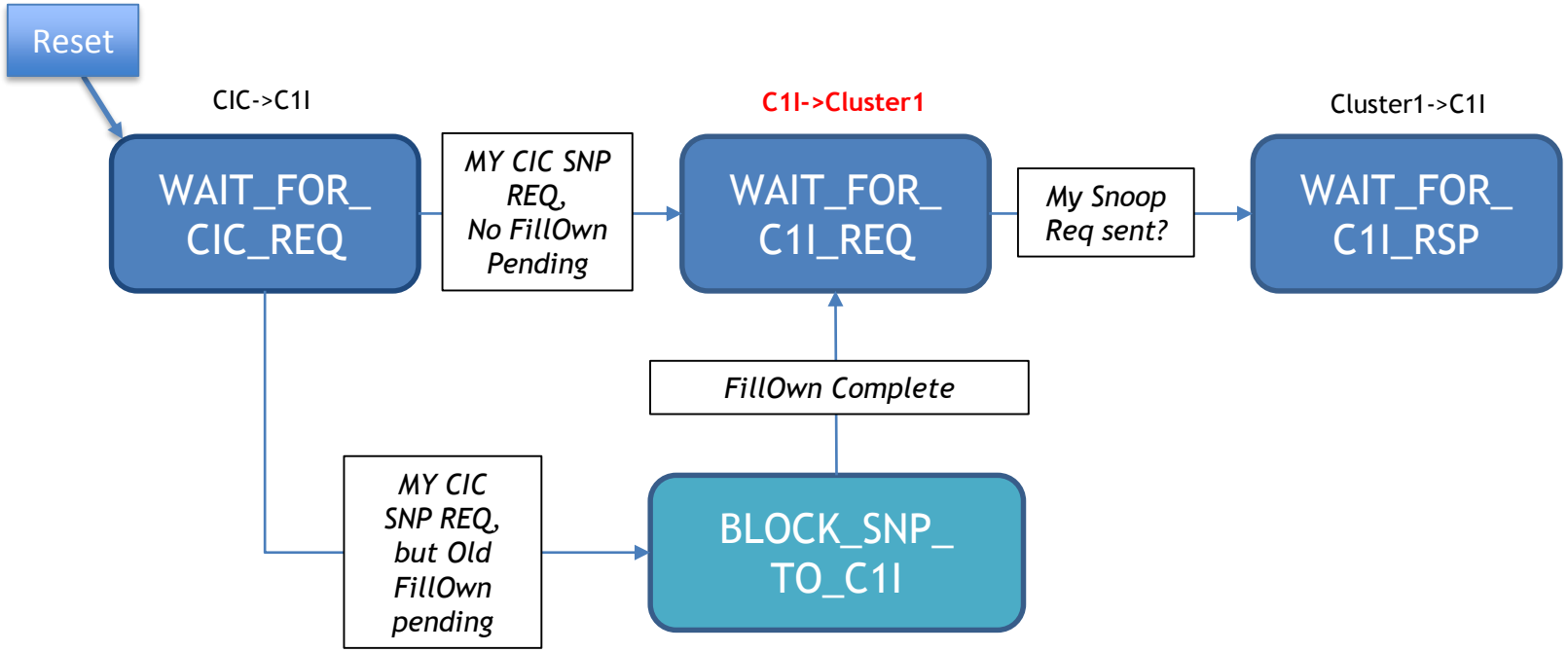
Snoop Tracker



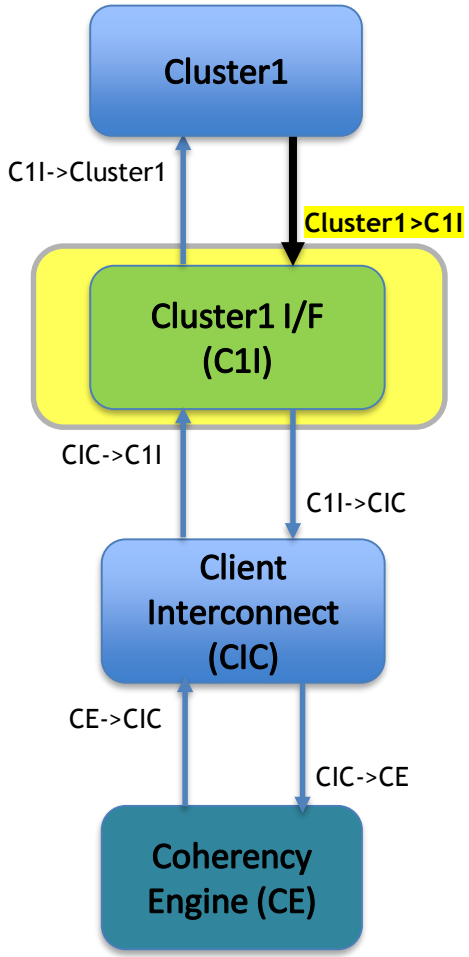


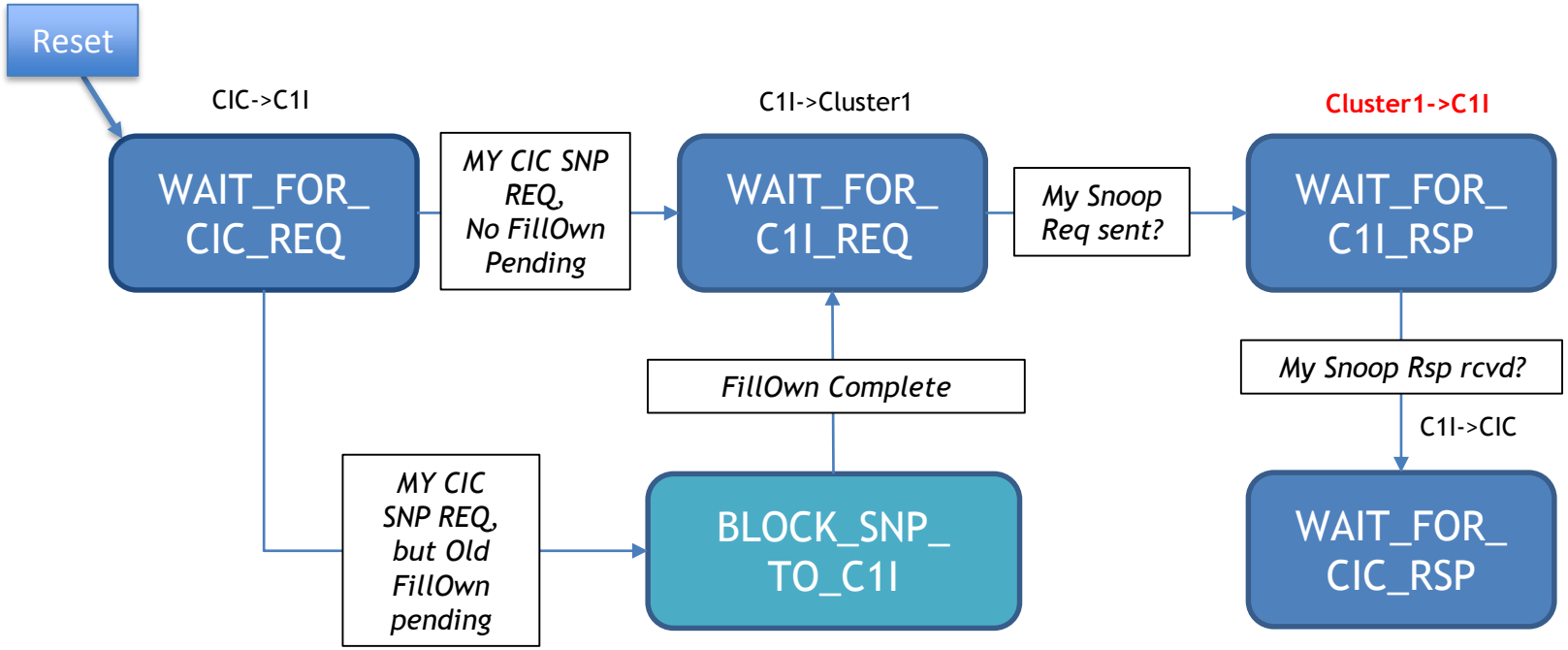
Snoop Tracker



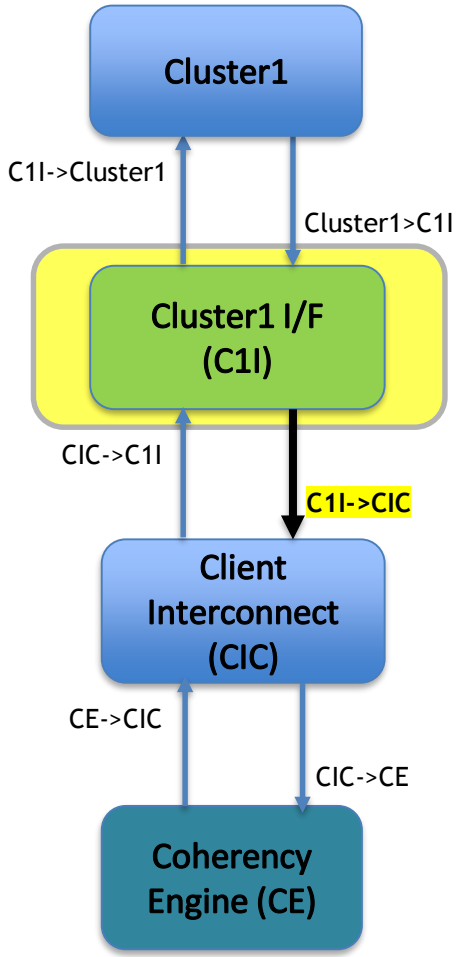


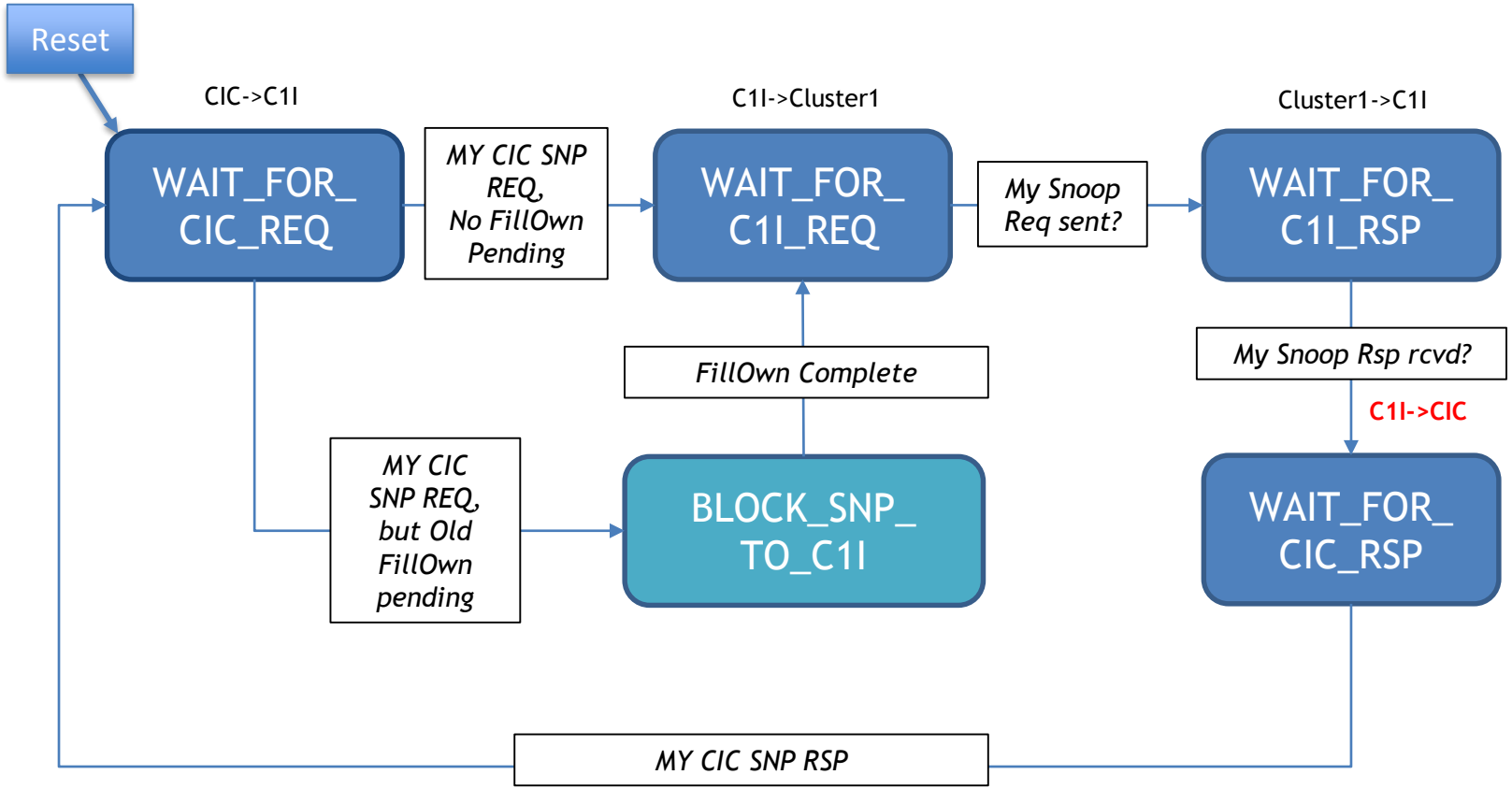
Snoop Tracker



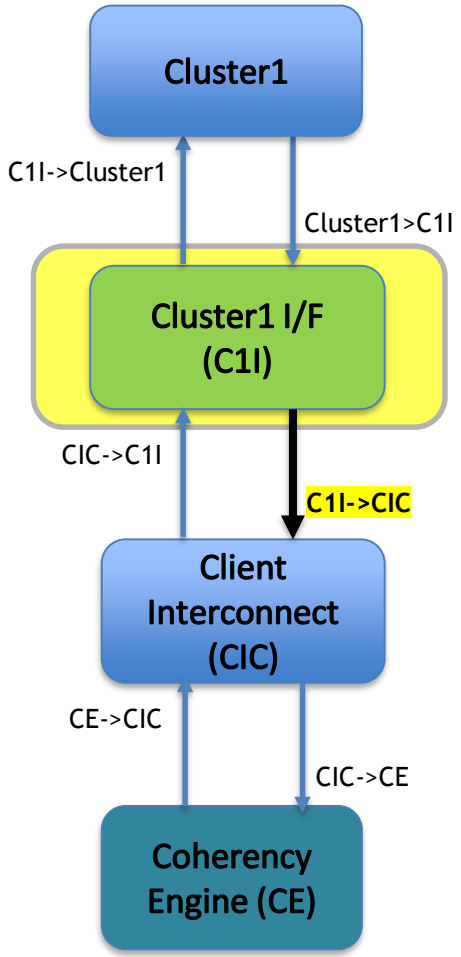


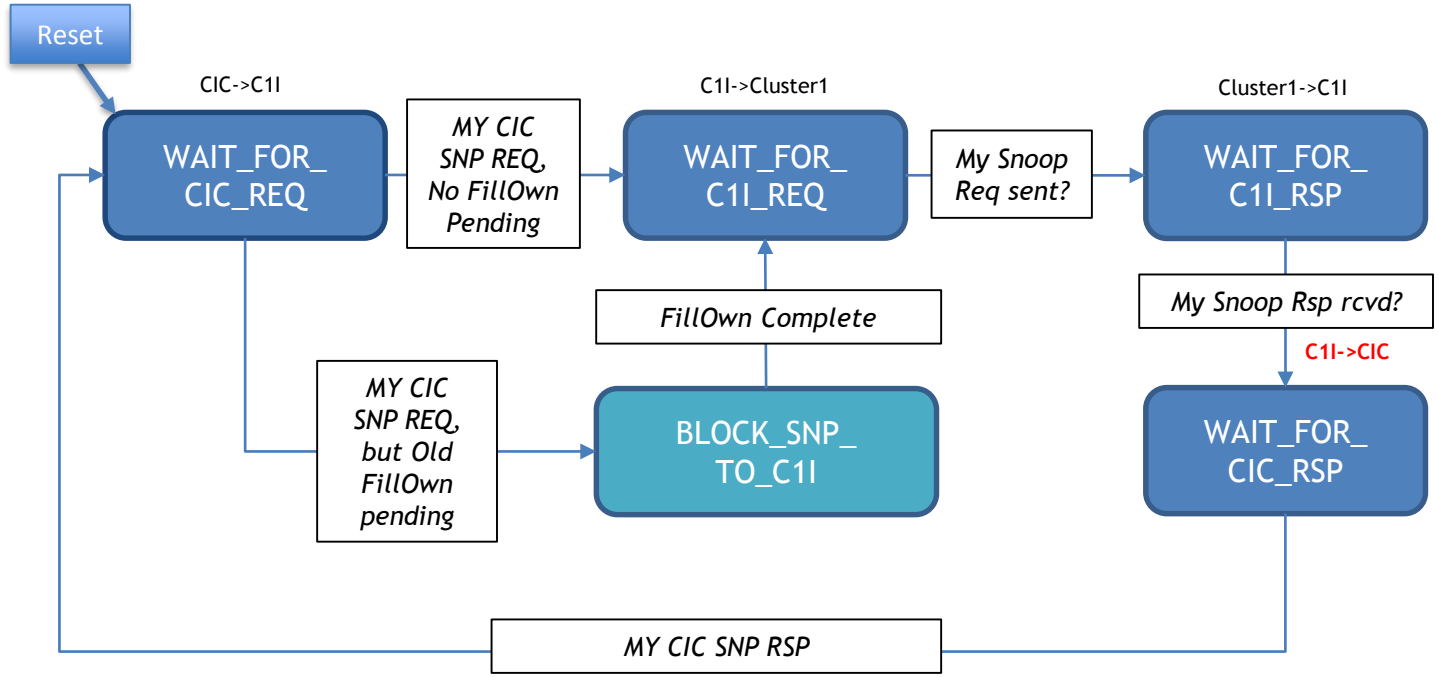
Snoop Tracker





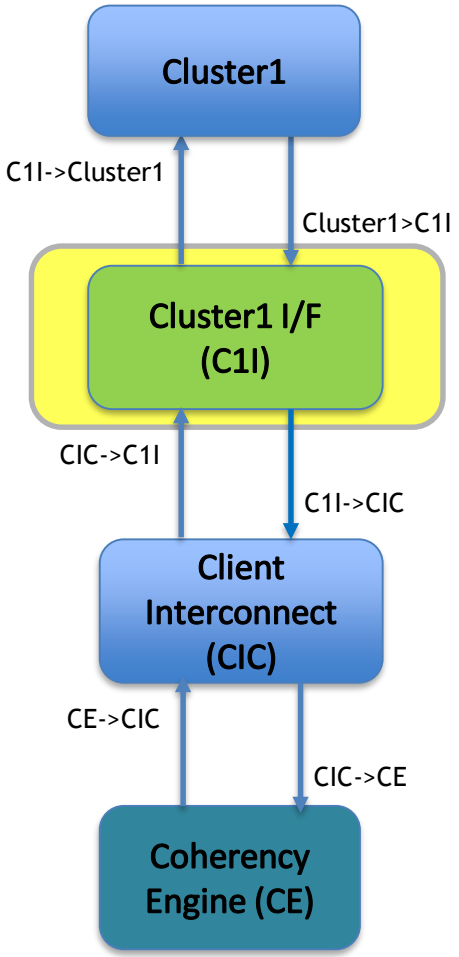
Snoop Tracker

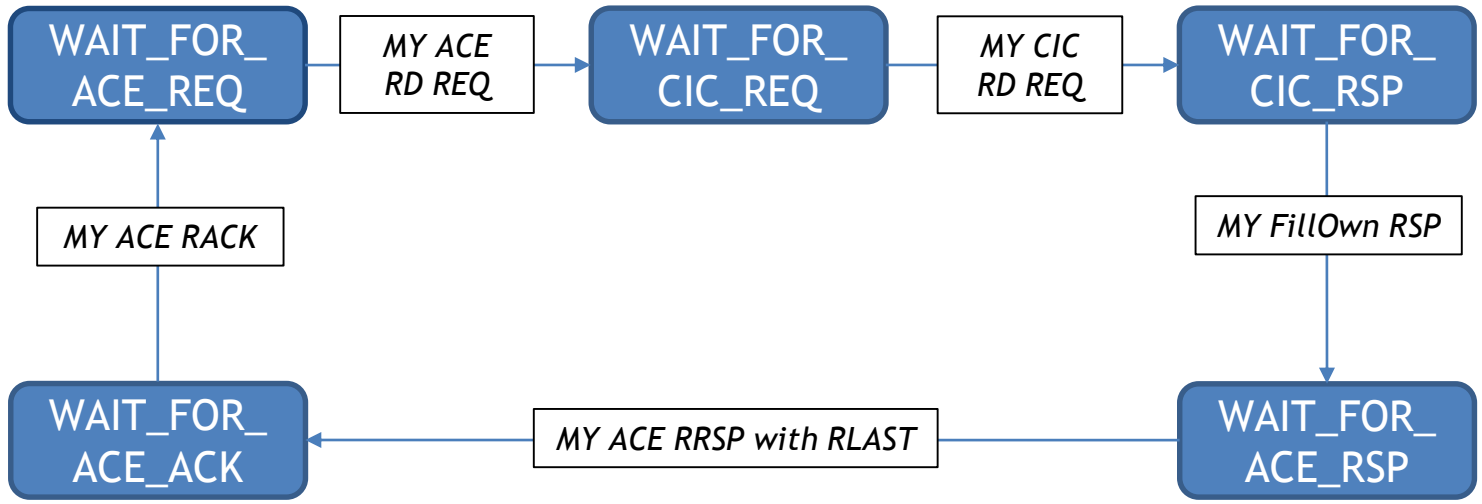




- Properties:
  - Final Snoop response must be as per original snoop request.
  - Snoop should push Fillown.

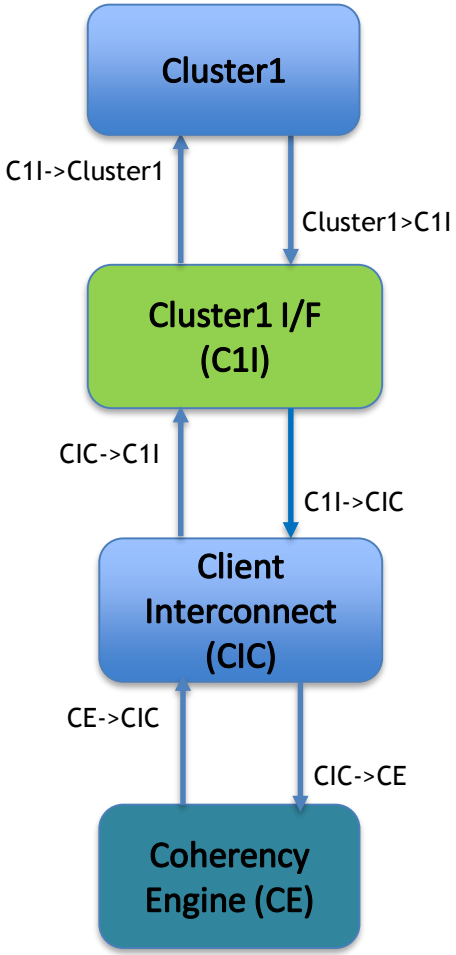
Read Tracker





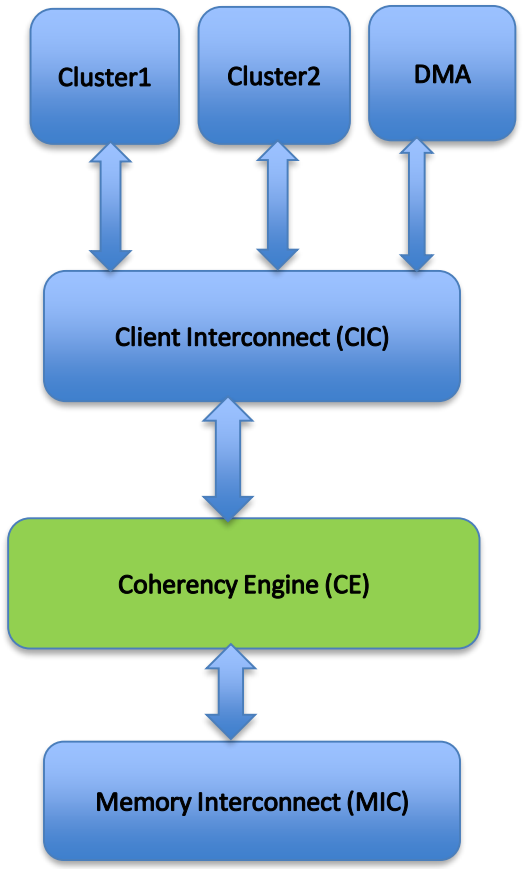
- Properties:
  - Read Request Consistency
  - Read Re-order buffer entry reuse
  - FIFO ordering rules on RRESP (on per ARID basis)
  - SoDev Ordering properties

C1I Properties



- Only 1 outstanding coherent request allowed for a cache-line
- If cache-line is not Unique, there should not be a dirty write back

Coherency Engine Architectural Model



- Snoop Filter
- Generates Proper FillOwn/WriteAck for each Read/Write request
- Models Full-Address Chain & Hazards
- Doesn’t Model: Data Values

Components of CE Architectural Model

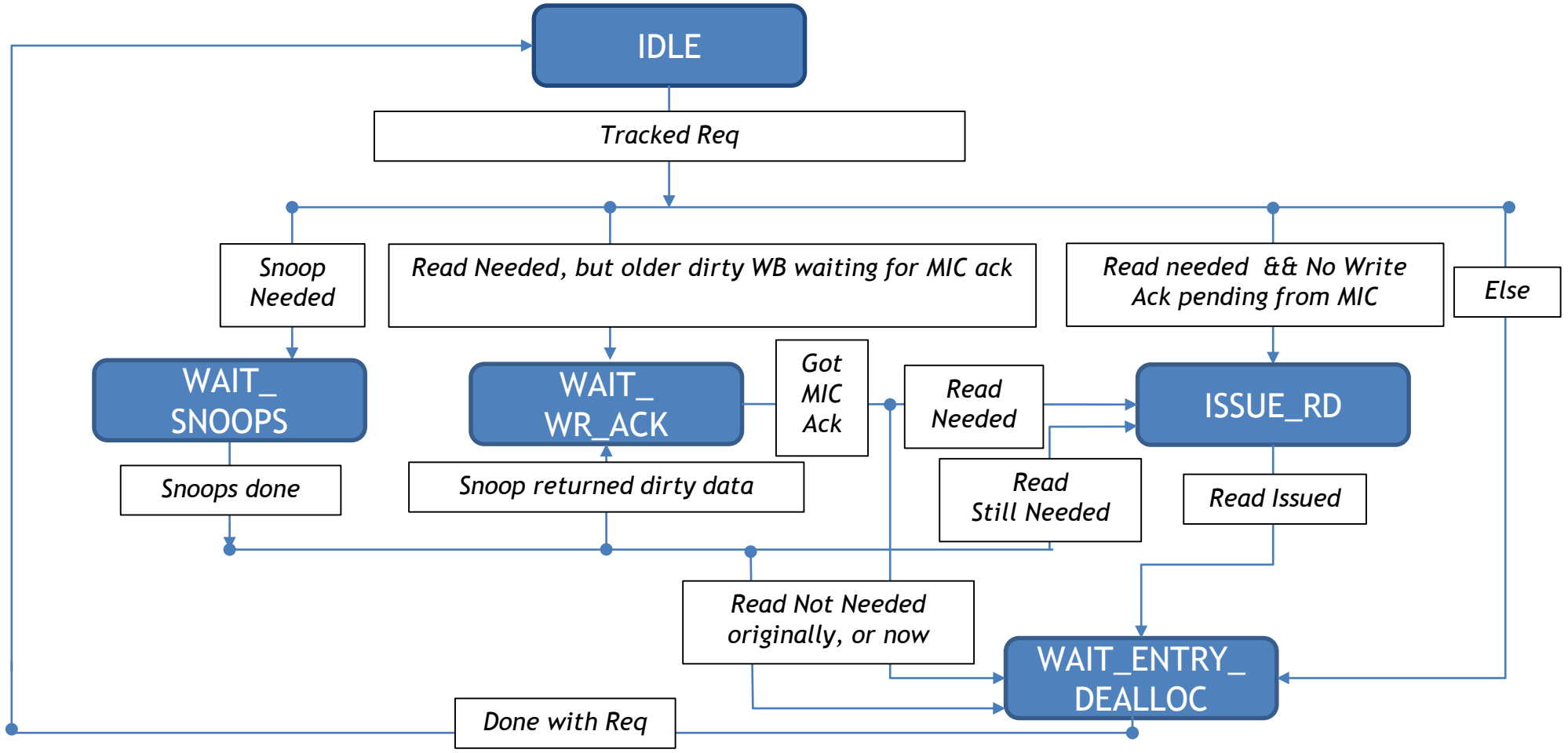
1. Read Request FIFO
   - Serialize read requests.
   - CE processes 1 read / address at a time.

2. Top-Of-FIFO State Machine
   - Models actions executed by CE to process a single read request / cacheline.

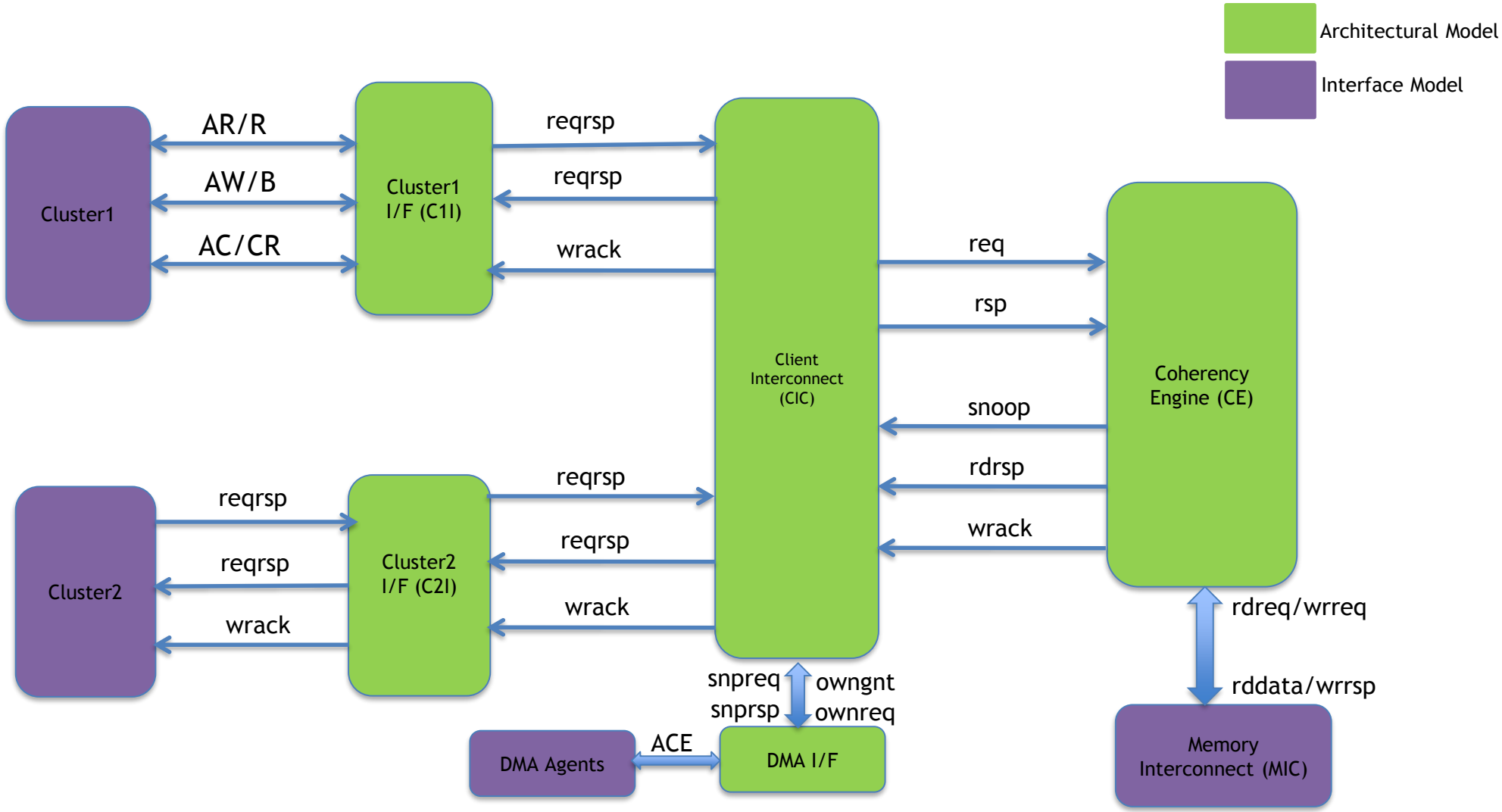
3. Snoop State Machine
   - Track outstanding snoops for tracked cacheline address.

4. Write Tracker: Tracks outstanding writebacks from agents.

Top-of-FIFO State Machine



CM Architectural Model



Results

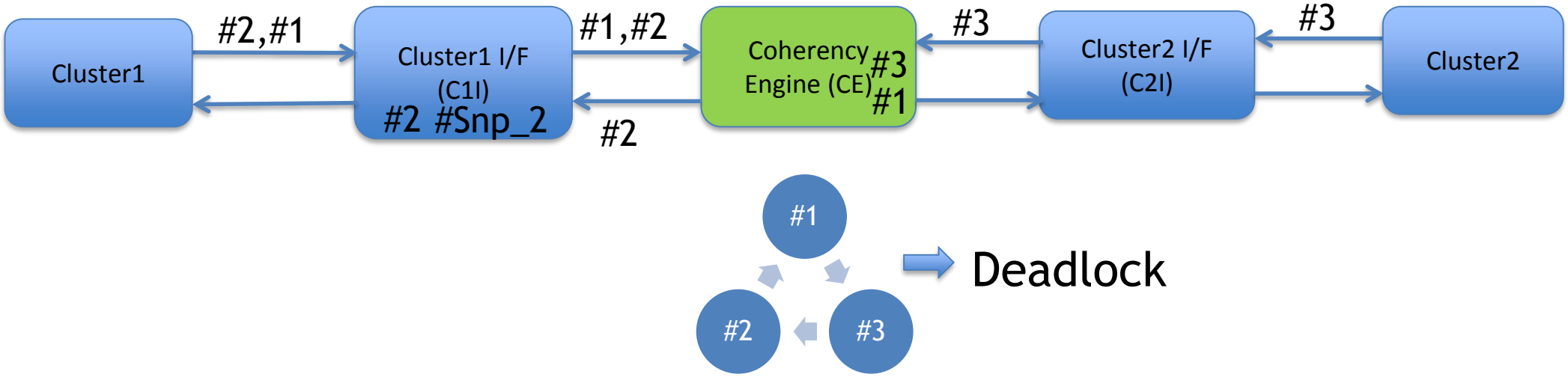
- Coherency Verification

```
$onehot0({    (cl_state_cluster1==Unique),
              (cl_state_cluster2==Unique),
              (cl_state_dma==Unique)    })
```

- Protocol Compliance

Bugs

- C1I re-orders Read requests with same AXI-ID from cluster1 to coherency engine (CE).



- C1I sends IsShared=1 for Failed STREX

Key Takeaways

- Architectural Formal Verification:
  - System level checking.
  - FV Applied at various abstraction levels.
    - Reduce Complexity
  - Prove local properties against RTL
  - Example use cases
    - Cache Coherency
    - Forward progression of retiring instructions

Acronyms

- CM: Coherency Manager
- AFV: Architectural Formal Verification
- C1I: Cluster1 Interface
- C2I: Cluster2 Interface
- DMA I/F: DMA Interface
- CIC: Client Interconnect
- MIC: Memory Interconnect
- CE: Coherency Engine

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q&A

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

• Introduction	: Sean Safarpour	(20 min)
• Induction & Invariants – Steps to Convergence	: Iain Singleton	(45 min)
• Efficient Formal Modeling Techniques	: Shaun Feng	(45 min)
Break		(10 min)
• Architectural Formal Verification for Coherency	: Syed Suhaib	(45 min)
• Formal Sign-off	: Mandar Munishwar	(45 min)

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Formal Sign-Off
What And How?

Mandar Munishwar
Sr. Staff Engineer, Qualcomm
Email: mmunishw@qti.qualcomm.com

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Outline

- Introduction
- What is Formal Sign-off
- Steps for Formal Sign-off
  - Plan
  - Execute
  - Measure
- Conclusion

Silicon Bugs …

OCT 1994

Pentium FDIV: The processor bug that shook the world

APR 2017

RISC-V bugs found by Princeton

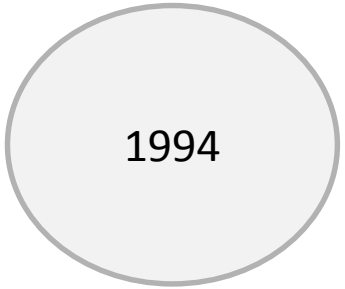
TriCheck, which has taken four years to develop, uses formal specifications of memory ordering – axioms.

JAN 2018

Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?

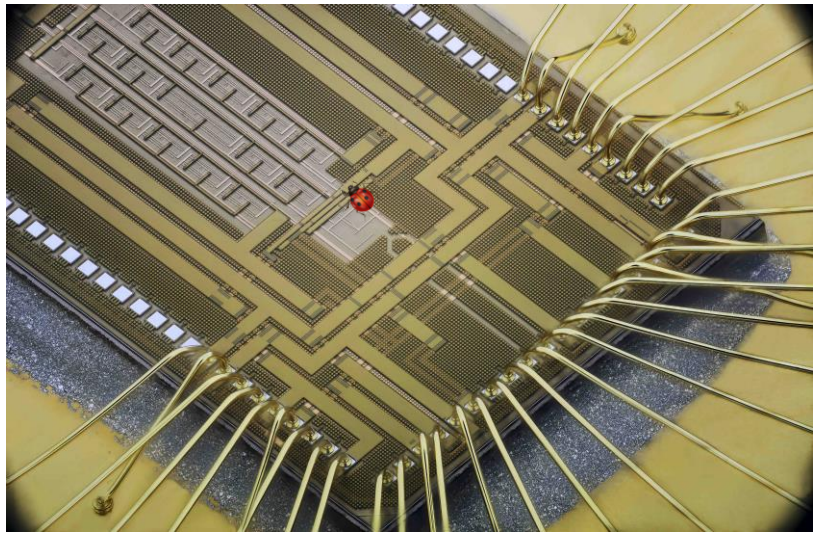
Why these escaped verification ?
Traditional Verification dependent on vectors/stimulus

Impact of Silicon Bugs



What is Formal Sign-Off

- Can my formal setup catch all the design bugs?

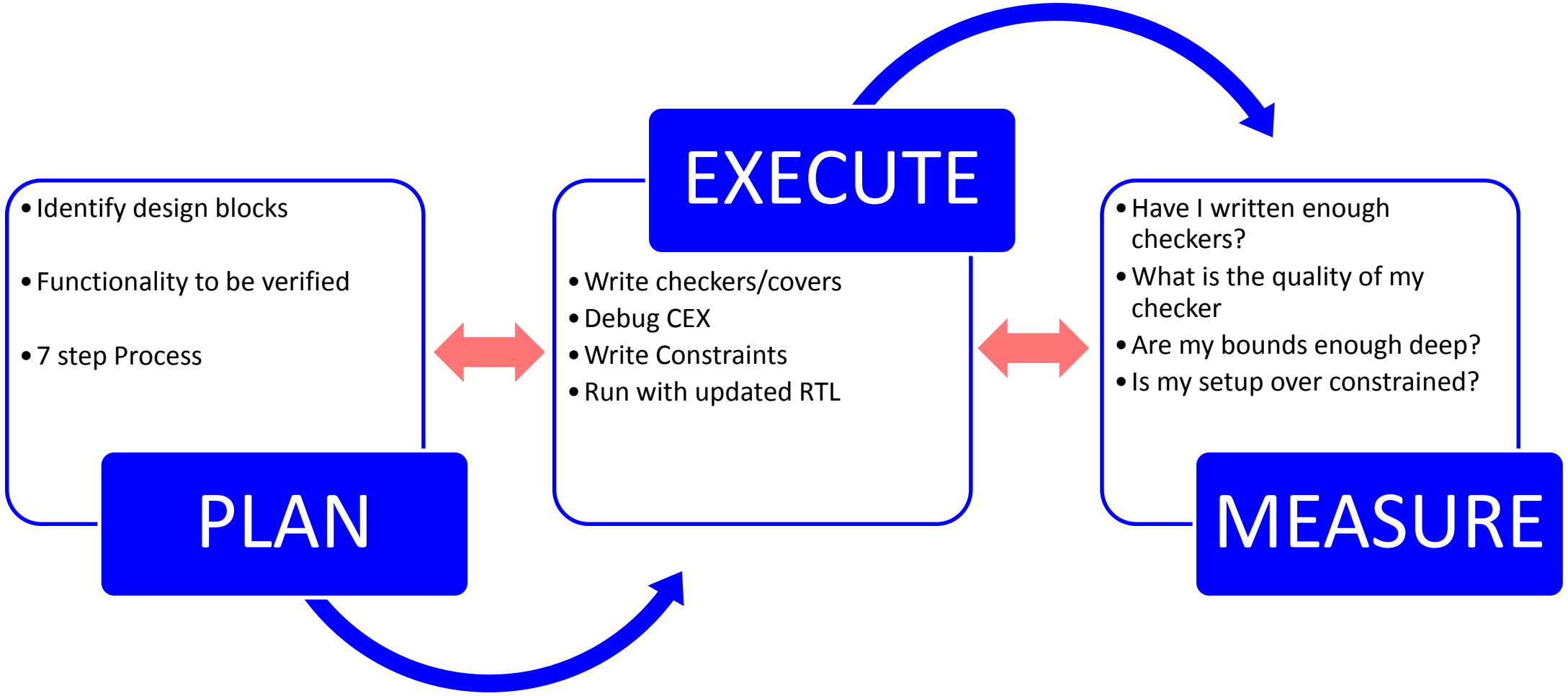


What is Formal Sign-Off

- Have I written all the checks 
- What is quality of checks? 
- Is there any Over constraints? 

Formal Verification Process

Front-loading vs. Back-loading



PLAN

1. Identify blocks suitable for Formal Sign-off
2. Capture Functional behavior
3. Define Formal Specification Interface
4. Create Requirements Checklist in Natural Language
5. Formalize Natural Language Requirements Checklist

Capture Functional Behavior (step 2)

- Shape of the signal
- Interface relationship
- Causality
- Forward Progress
- Signal Integrity (transport)

Example of Formal Specification Interface (step 3)

			Inputs	desc	Outputs	desc
[illegible]						
			[illegible]	[illegible]	[illegible]	[illegible]
[illegible]						
			[illegible]	[illegible]	[illegible]	[illegible]
			[illegible]	[illegible]	[illegible]	[illegible]
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]	[illegible]	[illegible]	[illegible]
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
			[illegible]		[illegible]	
[illegible]						
			[illegible]		[illegible]	[illegible]
			[illegible]		[illegible]	[illegible]
			[illegible]	[illegible]	[illegible]	

Example of step 4, 5

Checkers

Interface Name	Outputs	Desc	SVA	STATUS	Note
SCHD2BMMU					
	bmmu_gnt	signal is a pulse	bmmu2schd_bmmu_gnt_is_pulse_a		
			bmmu2schd_bmmu_no_gnt_if_no_req_a		
		for each req, bmmu should provide gnt within N cycles	bmmu2schd_bmmu_rst_gnt_forward_progress_chk_a		
			bmmu2schd_bmmu_ini_gnt_forward_progress_chk_a		
			bmmu2schd_bmmu_pop_gnt_fwd_progress_chk20_a		
			bmmu2schd_bmmu_dlt_gnt_forward_progress_chk_a		
			bmmu2schd_bmmu_pwrdn_gnt_forward_progress_chk_a		

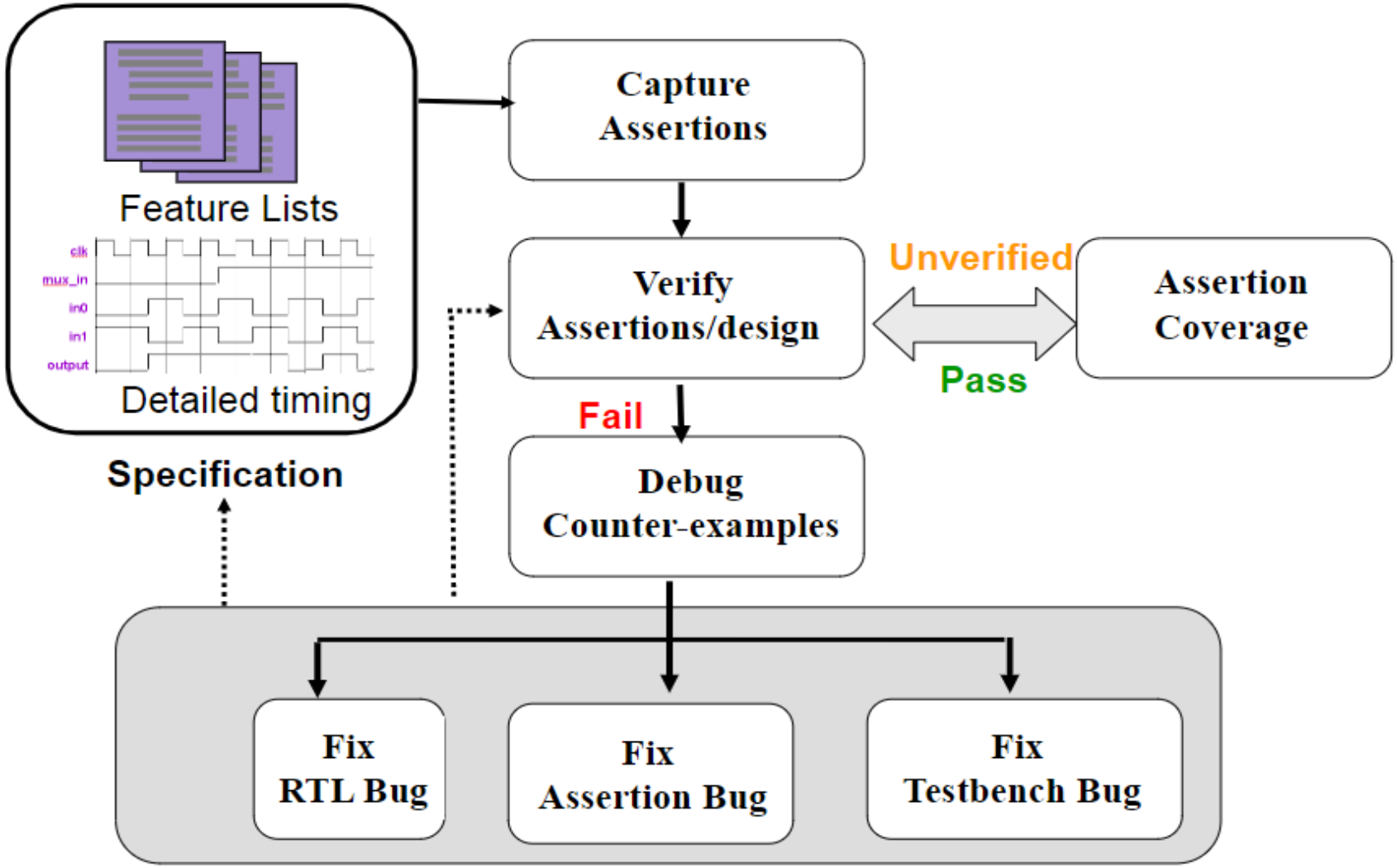
Constraints

Interface name	signals	Constraints	SVA
deint interface x 4			
	llr_dat		
	llr_valid	no valid for 15 cycles after last init gnted by bmmu	deint2bmmu_no_data_for_15cyc_after_last_init_gnted_c
	llr_usr	UID never out of range (less than 20 per bank)	deint2bmmu_no_uid_oor_c
		forbid invalid uid (uid that is not initialized)	deint2bmmu_forbid_invalid_deint_llr_usr_c
		same uid cannot be on multiple channels in a given cycle (per uid)	deint2bmmu_no_duplicate_uid_across_chnl_c

Checks for common design components

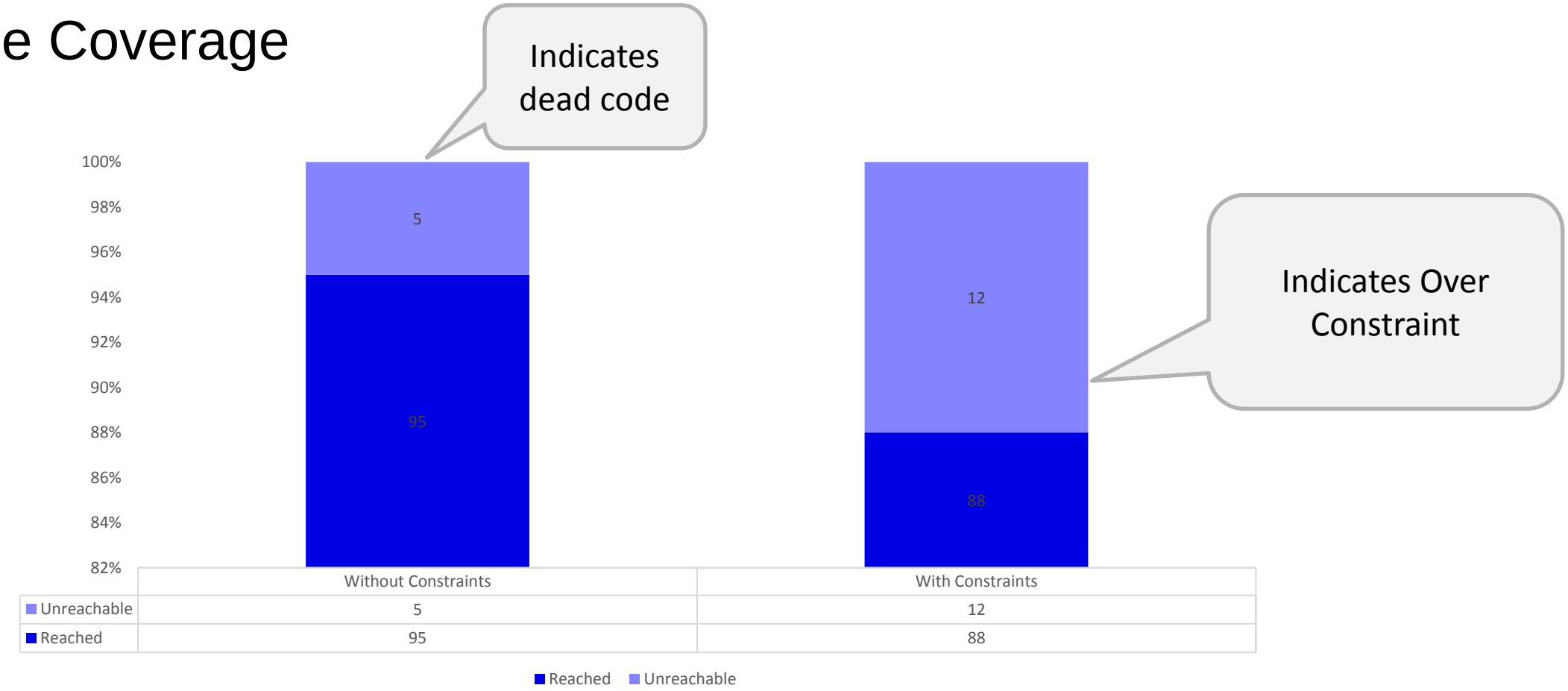
- FIFOs
- Counters
- Arbiters

EXECUTE


Feature Lists
clk
mux_in
in0
in1
output
Detailed timing
Specification
Capture Assertions
Verify Assertions/design
Unverified
Assertion Coverage
Pass
Fail
Debug Counter-examples
Fix RTL Bug
Fix Assertion Bug
Fix Testbench Bug

MEASURE
Is my setup over constrained?

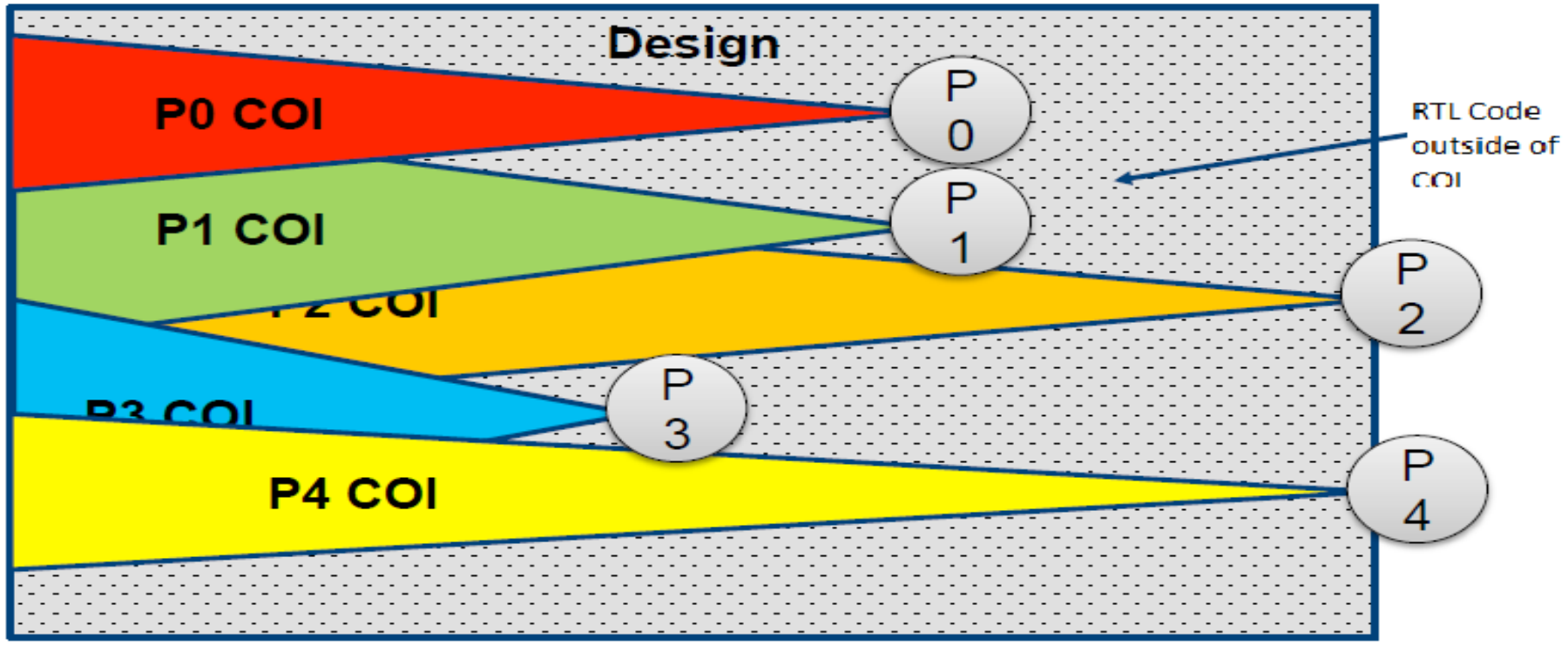
- Code Coverage



MEASURE

Have I written enough checkers?

- COI Coverage
- < 100% COI Coverage indicates missing checkers

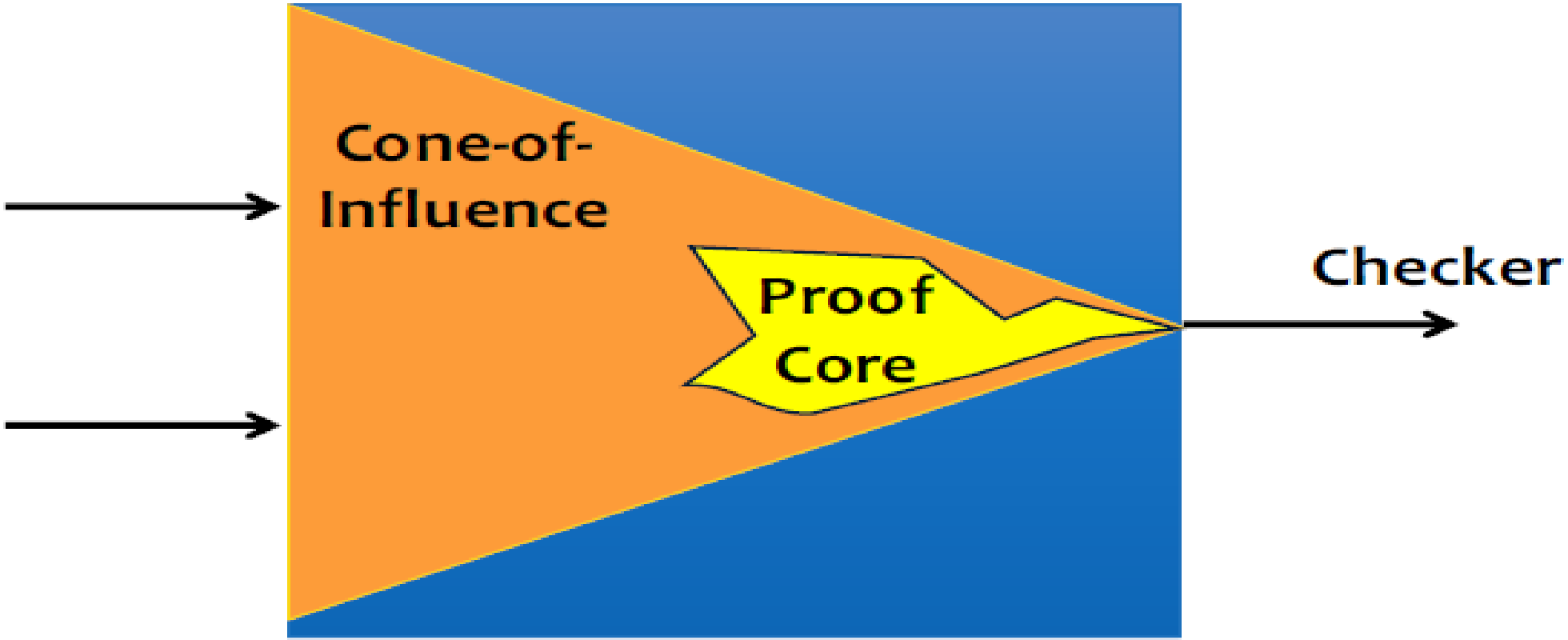


MEASURE
Quality of checkers

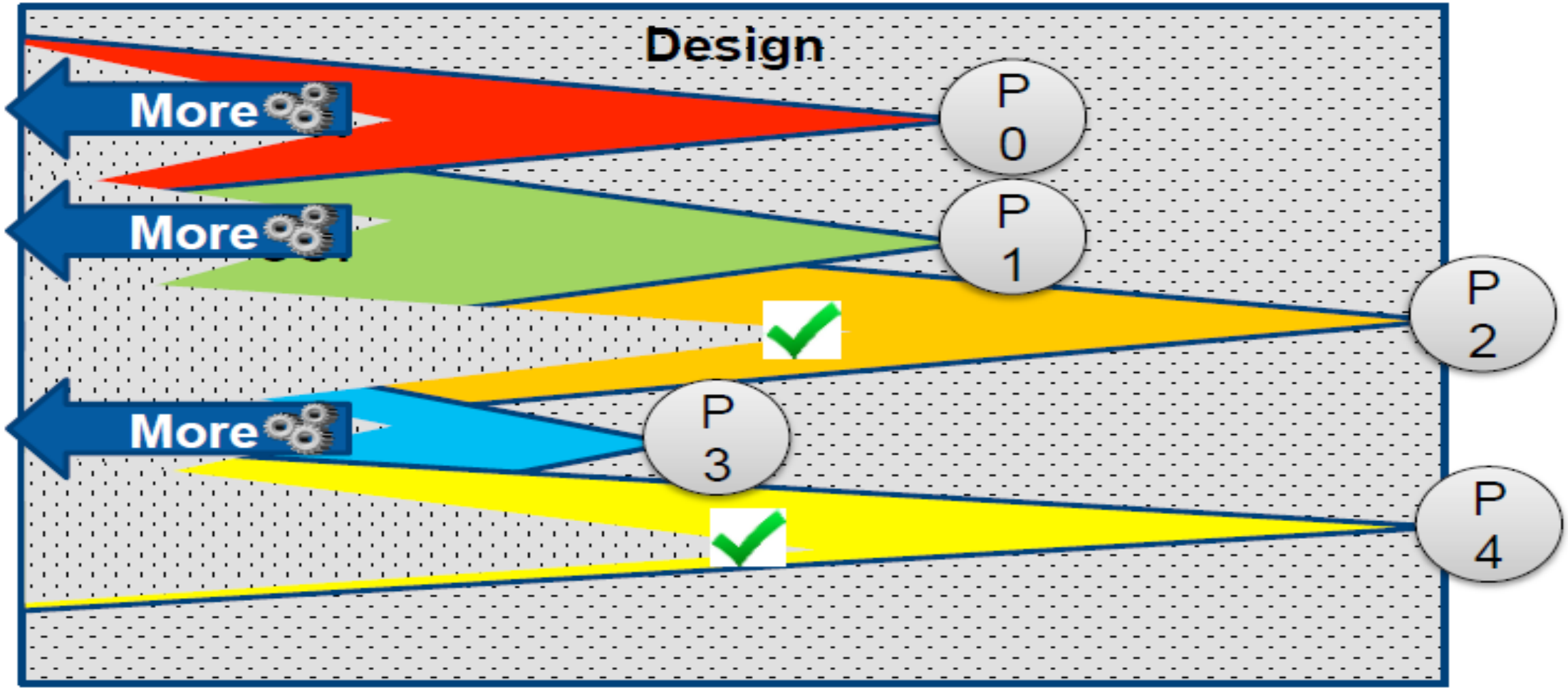
- Subjective
- A good job at planning phase will ensure quality
- Diversified checkers
- Technical review with team

MEASURE
Quantifying the Quality of checkers

- Formal Core

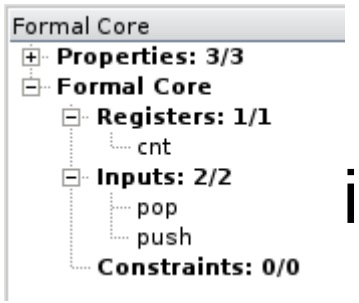


Formal-core Coverage

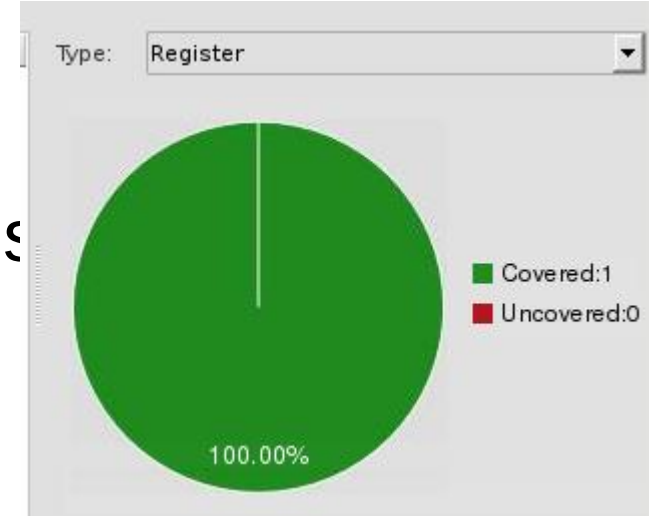


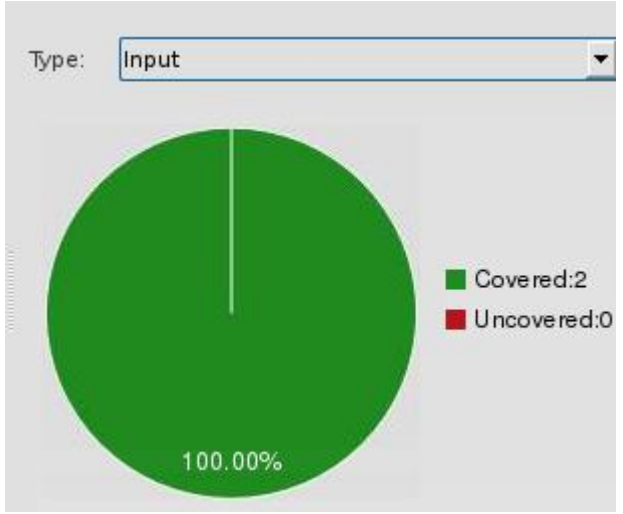
100% Formal-core Coverage

```
module top(input clk,
           input rst_x,
           input push,
           input pop,
           output logic full,
           output logic empty);

   logic [3:0] cnt;

   always@(posedge clk or negedge rst_x) begin
     if(!rst_x) begin
       cnt <= '0;
     end else begin
        if(push && ~pop && cnt!=4'hf) begin
          cnt <= cnt + 1'b1;
        end else if (!push && pop && cnt!= 4'h0) begin
          cnt <= cnt - 1'b1;
        end else begin
          cnt <= cnt;
        end
     end
   end //always

   assign full = (cnt == 4'hf);
   assign empty = (cnt == 4'h0);

   P1: assert property ( @(posedge clk) cnt == 4'hf |-> full);
   P2: assert property ( @(posedge clk) cnt == 4'h0 |-> empty);
   P3: assert property ( @(posedge clk) ~push&~pop |=> $stable(cnt));
```



is

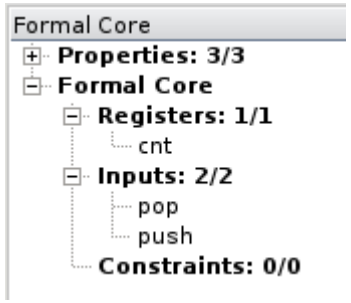


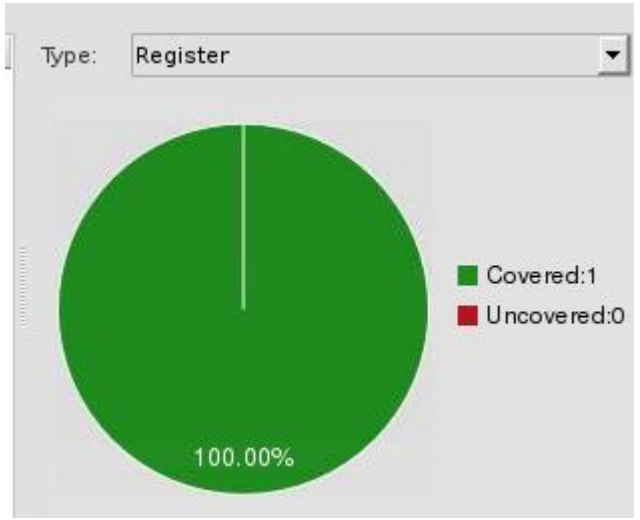


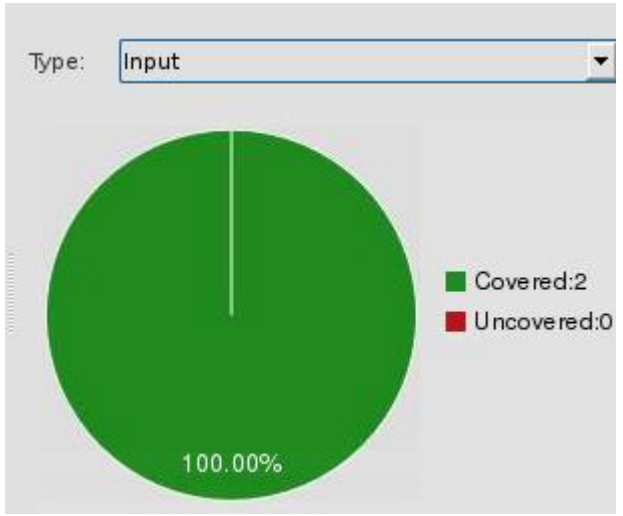

```
        cnt <= '0;
      end else begin
         if(push && ~pop && cnt!=4'hf) begin
           cnt <= cnt + 1'b1;
         end else if (!push && pop && cnt!= 4'h0) begin
           cnt <= cnt - 1'b1;
         end else begin
           cnt <= cnt;
         end
      end
    end //always
```

Let's introduce one …

```
module top(input clk,
           input rst_x,
           input push,
           input pop,
           output logic full,
           output logic empty);

  logic [3:0] cnt;

  always@(posedge clk or negedge rst_x) begin
    if(!rst_x) begin
      cnt <= '0;
    end else begin
      if(push && ~pop && cnt!=4'hf) begin
        cnt <= cnt + 1'b2;
      end else if (!push && pop && cnt!= 4'h0) begin
        cnt <= cnt - 1'b1;
      end else begin
        cnt <= cnt;
      end
    end
  end //always

  assign full = (cnt == 4'hf);
  assign empty = (cnt == 4'h0);

  P1: assert property ( @(posedge clk) cnt == 4'hf |-> full);
  P2: assert property ( @(posedge clk) cnt == 4'h0 |-> empty);
  P3: assert property ( @(posedge clk) ~push&~pop |=> $stable(cnt));
```







	status	depth	name	vacuity	witness	engine	type
1	✔		top.P1			t3	assert
2	✔		top.P2			t3	assert
3	✔		top.P3			e1	assert

- My checkers are still passing

Formal-core Coverage

- As with any other structural coverage, 100 % formal-core coverage does not mean much

- What are the options?

WELCOME to MUTATION

What is Mutation?

- Modifying the DUT in small ways
- Can this modification be detected by checkers?

Original

```
if (a && b) {
    c = 1;
} else {
    c = 0;
}
```

Mutant

```
if (a || b) {
    c = 1;
} else {
    c = 0;
}
```

Applying Mutation – 1st Iteration

```
module top(input clk,
           input rst_x,
           input push,
           input pop,
           output logic full,
           output logic empty);

   logic [3:0] cnt;

   always@(posedge clk or negedge rst_x) begin
     if(!rst_x) begin
       cnt <= '0;
     end else begin
        if(push && ~pop && cnt!=4'hf) begin
          cnt <= cnt + 1'b1;
        end else if (!push && pop && cnt!= 4'h0) beg
          cnt <= cnt - 1'b1;
        end else begin
          cnt <= cnt;
        end
     end
   end //always

   assign full = (cnt == 4'hf);
   assign empty = (cnt == 4'h0);

   P1: assert property ( @(posedge clk) cnt == 4'hf |-> full);
   P2: assert property ( @(posedge clk) cnt == 4'h0 |-> empty);
   P3: assert property ( @(posedge clk) ~push&~pop |=> $stable(cnt));
```

Class Name	Faults in Design	Faults in Report	Non-Activated	Detected	Non-Detected	Disabled By User
TopOutputsConnectivity	0	0	0	0	0	0
ResetConditionTrue	1	1	0	0	1	0
SynchronousControlFlow	8	8	0	4	4	0
InternalConnectivity	0	0	0	0	0	0
SynchronousDeadAssign	0	0	0	0	0	0
ComboLogicControlFlow	0	0	0	0	0	0
SynchronousLogic	15	15	0	4	11	0
ComboLogic	10	10	0	10	0	0
OtherFaults	1	1	0	0	1	0

```
Properties: 35 - passed[18] - failed[17] - disabled[0];
```

Applying Mutation – 2nd Iteration

```
module top(input clk,
           input rst_x,
           input push,
           input pop,
           output logic full,
           output logic empty);

  logic [3:0] cnt;

  always@(posedge clk or negedge rst_x) begin
    if(!rst_x) begin
      cnt <= '0;
    end else begin
       if(push && ~pop && cnt!=4'hf) begin
         cnt <= cnt + 1'b1;
       end else if (!push && pop && cnt!= 4'h0) begin
         cnt <= cnt - 1'b1;
       end else begin
         cnt <= cnt;
       end
    end
  end //always

  assign full = (cnt == 4'hf);
  assign empty = (cnt == 4'h0);

  P1: assert property ( @(posedge clk) cnt == 4'hf |-> full);
  P2: assert property ( @(posedge clk) cnt == 4'h0 |-> empty);
  P3: assert property ( @(posedge clk) ~push&~pop |=> $stable(cnt));
  P4: assert property ( @(posedge clk) push&~pop && (cnt!=4'hf) |=> (cnt == $past(cnt) +1));
  P5: assert property ( @(posedge clk) ~push&pop && (cnt!=4'h0) |=> (cnt == $past(cnt) -1));
```

Class Name	Faults in Design	Faults in Report	Non-Activated	Detected	Non-Detected	Disabled By User
TopOutputsConnectivity	0	0	0	0	0	0
ResetConditionTrue	1	1	0	1	0	0
SynchronousControlFlow	8	8	0	7	1	0
InternalConnectivity	0	0	0	0	0	0
SynchronousDeadAssign	0	0	0	0	0	0
ComboLogicControlFlow	0	0	0	0	0	0
SynchronousLogic	15	15	0	14	1	0
ComboLogic	10	10	0	10	0	0
OtherFaults	1	1	0	1	0	0

Properties: 35 - passed[33] - failed[2] - disabled[0]

What Are The Ways To Mutate?

- Statement deletion
- Statement duplication or insertion, e.g. `goto fail;` [15]
- Replacement of boolean subexpressions with *true* and *false*
- Replacement of some arithmetic operations with others, e.g. `+` with `*`, `-` with `/`
- Replacement of some boolean relations with others, e.g. `>` with `>=`, `==` and `<=`
- Replacement of variables with others from the same scope (variable types must be compatible)

Mutant Classification

- Top Outputs Connectivity
- Reset Condition True
- Internal Connectivity
- Synchronous Flow Control
- Synchronous Dead Assign
- Combo Logic Control Flow

Example of TopOutput Connectivity Faults

- module topMod (output out1…);

- assign out1 = (opsa0en == 1'b1) ? ('0) : // OutputPortStuckAt0
- (opsa1en == 1'b1) ? ('1) : // OutputPortStuckAt1
- (opnegen == 1'b1) ? (~orig_out1) : // OutputPortNegated
- orig_out1 ;
- …
- endmodule

Example of ResetCondition True Fault

```
always @(posedge clk or negedge rstn) begin
        if (!rstn) begin
          ….
        end else …
      ….
always @(posedge clk or negedge rstn) begin
        if (1’b1) begin
          ….
        end else …
      ….
```

Original

Mutated

Summary

- A well executed and measured plan can take us to the goal of Formal sign-off
- Plan
  - Well Defined process with diversified checkers identified
- Execution
  - All checkers passing or acceptable bounded depth
- Measurement
  - No over constraint
  - 100% Formal Core
  - Extra confidence with Mutation analysis

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Thank You