Deep Bug Hunting

- Asa Ben-Tzur, SW Group Director, Cadence

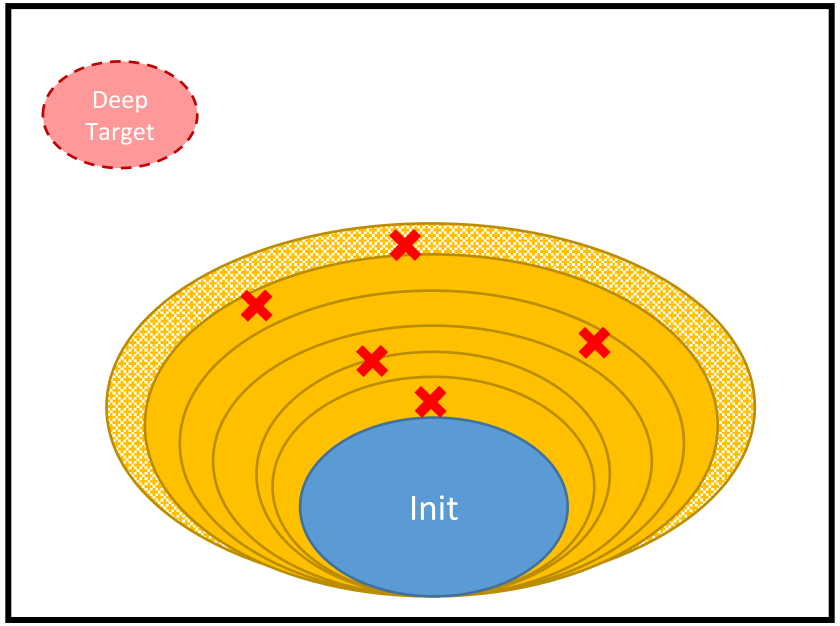
- Formal Verification Conference, Reading, UK
- 16th June 2016

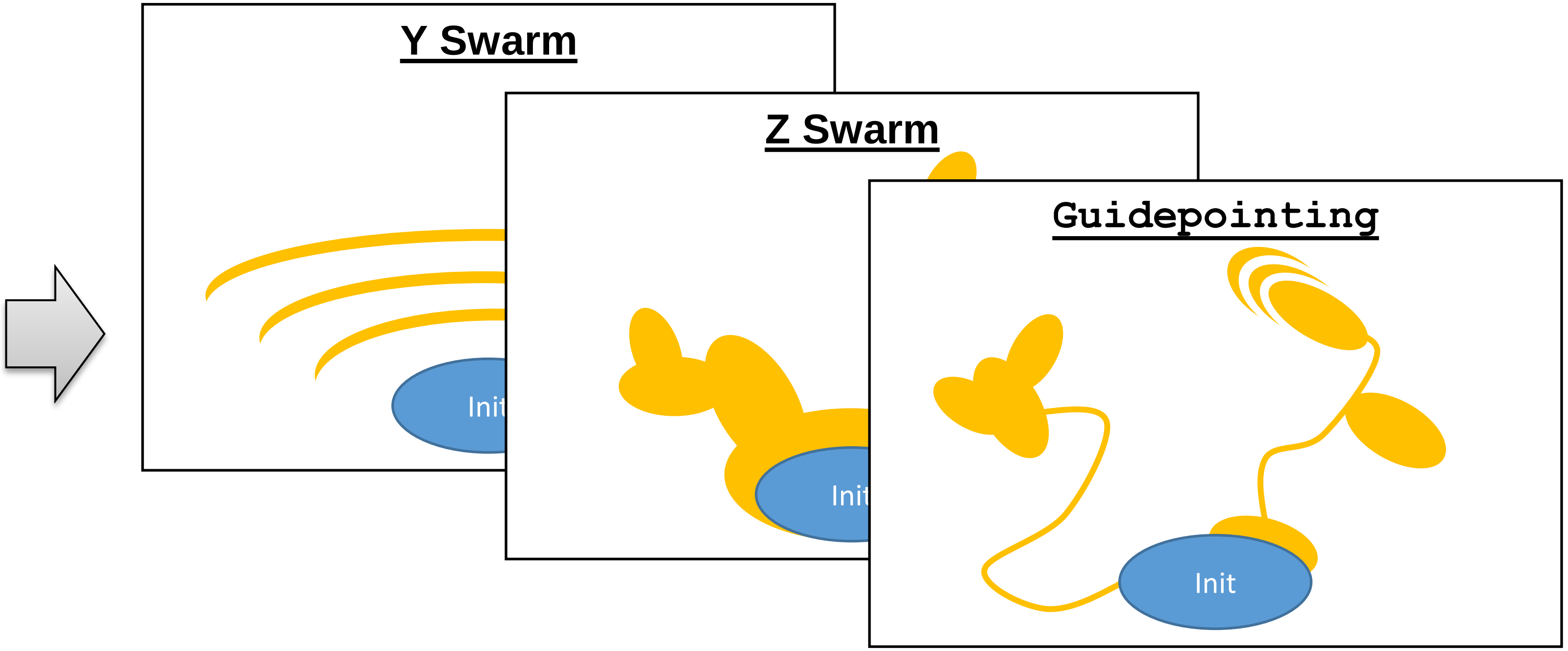
Agenda

- Introduction to Bug-Hunting
- Y Swarm
- Z Swarm
- Guidepointing

About Bug-Hunting

- Classic formal focuses on exhaustively analyzing the state-space
  - Wide search, thorough analysis
- Bug-Hunting focuses on finding bugs ☺
  - Deep/sparse search on state-space (not exhaustive)

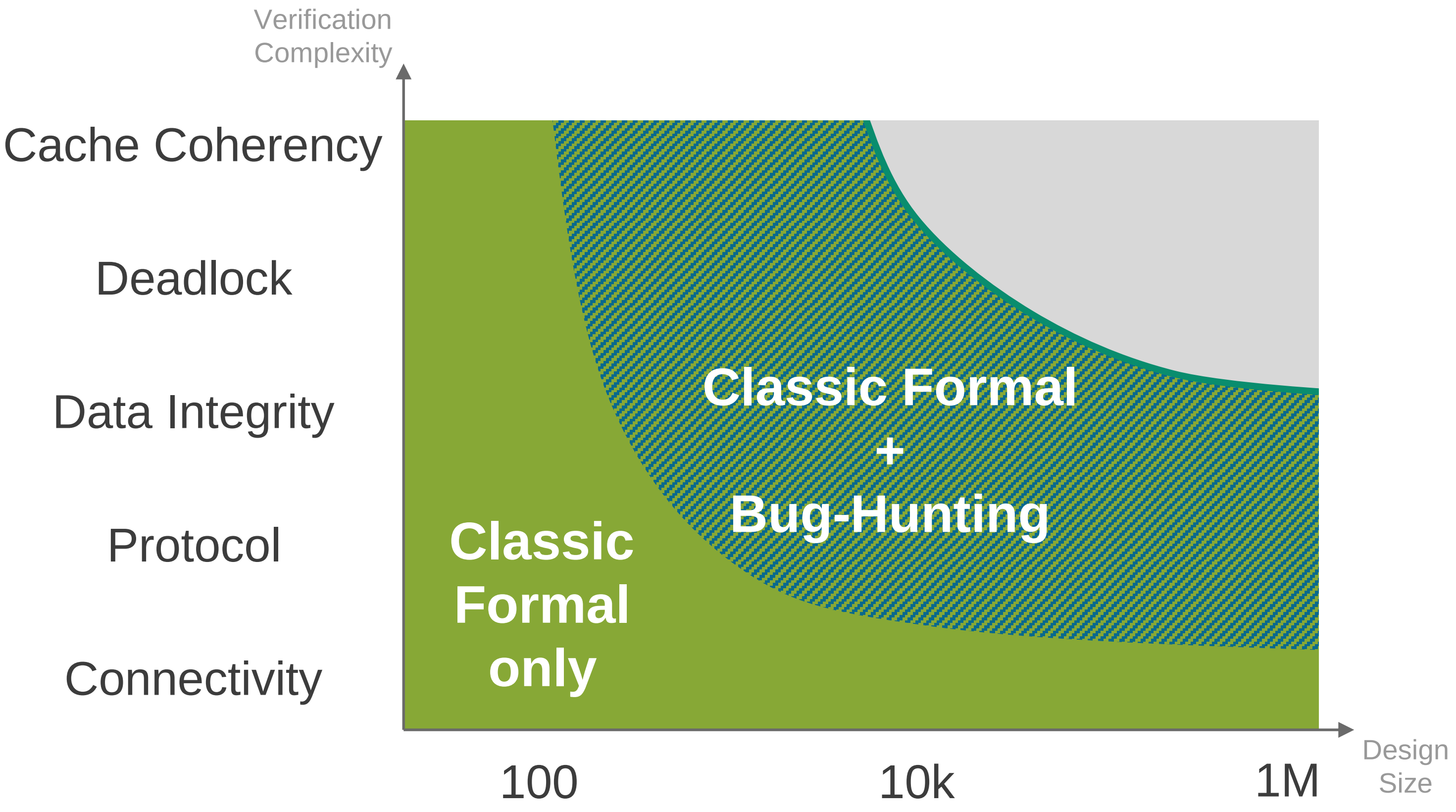




cadence

Where to use Bug-Hunting

- Most formal TBs can benefit from bug-hunting
- No need for bug-hunting on small designs or simple verification tasks



When to use Bug-Hunting

- Start with normal formal TB development
  - Develop assertions
  - Fix assertions
  - Find bugs using classic formal engines
  - Apply (simple) mutations/abstractions
- Move to bug-hunting when you:
  1. Have enough assertions (from coverage reports)
  2. Stop finding bugs - **clean environment**
- Explore more states with bug-hunting
  - Find bugs using semi-formal techniques/engines
  - Techniques build on top of existing formal TB

Find bugs using classic formal engines

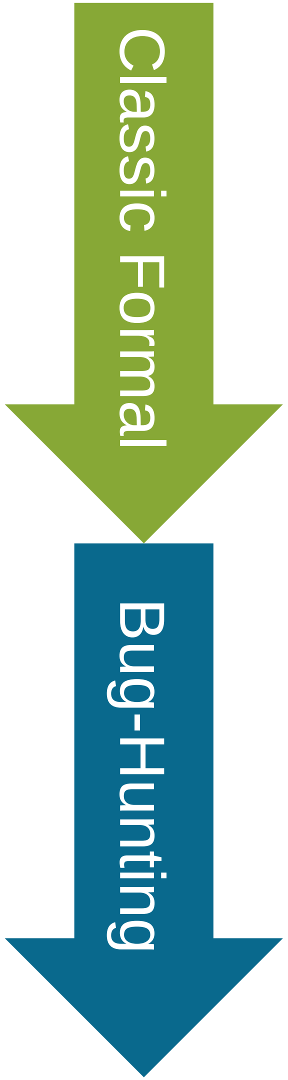
	Type	Name	Engine	Bound
?	Assert	AST_deq_start_sop		
✓	Assert	AST_deq_single_sop		
✗	Assert	AST_deq_first	Ht	10
✗	Assert	AST_deq_last	N	6
?	Assert	AST_no_double_alloc		
✗	Assert	AST_pop_not_empty	Hts	8
?	Assert	AST_push_not_full		

	Type	Name	Engine	Bound
?	Assert	AST_deq_start_sop		
✓	Assert	AST_deq_single_sop		
?	Assert	AST_deq_first		
✓	Assert	AST_deq_last		
?	Assert	AST_no_double_alloc		
?	Assert	AST_pop_not_empty		
?	Assert	AST_push_not_full		

Clean Environment

	Type	Name	Engine	Bound
?	Assert	AST_deq_start_sop		
✓	Assert	AST_deq_single_sop		
?	Assert	AST_deq_first		
✓	Assert	AST_deq_last		
✗	Assert	AST_no_double_alloc	B	1 - 16
?	Assert	AST_pop_not_empty		
✗	Assert	AST_push_not_full	L	1 - 184

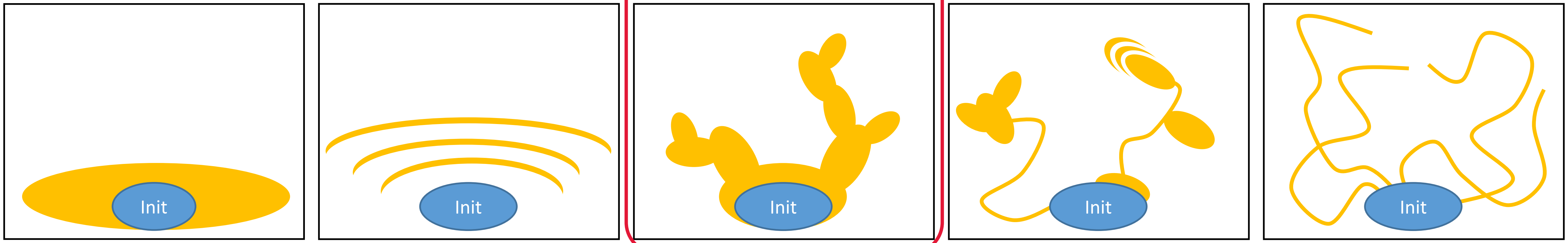
Find deep bugs using semi-formal techniques



cadence

Bug-Hunting Strategies

Classic Formal	**Y Swarm**	**Z Swarm**	**Guidepointing**	**Simulation**
Exhaustive analyze	Skip or only partially analyze areas in state space	Automatically guide formal search using key states	Manually guide analysis using formal and simulation	Constrained-random walk through states



Bug-Hunting workhorse!

Formal	Bug-Hunting (Semi-Formal)	Simulation

Agenda

- Introduction to Bug-Hunting
- **Y Swarm**
- Z Swarm
- Guidepointing

Y Swarm
Skip or only partially analyze state space

Formal	Bug-Hunting (Semi-Formal)	Simulation

Y Swarm

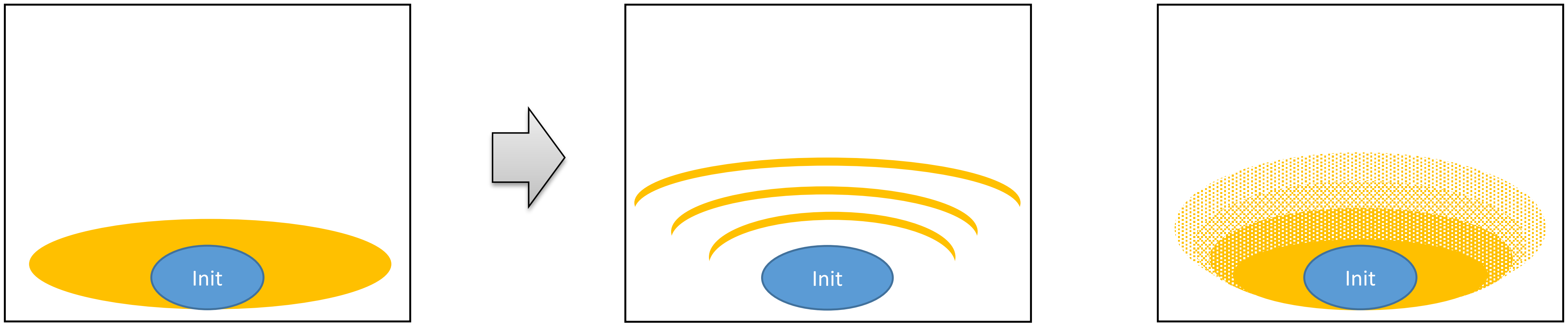
- Classic Formal performs thorough state space search
- Search can be configured to **skip/partially analyze state space areas**
  - **Y Swarm mode**
  - Not exhaustive, but goes deeper
- Enable massive parallel search
  - Customers utilize 10s-1000s parallel jobs in single session

Y Swarm mode

Default Search
exhaustive analyze

skip areas

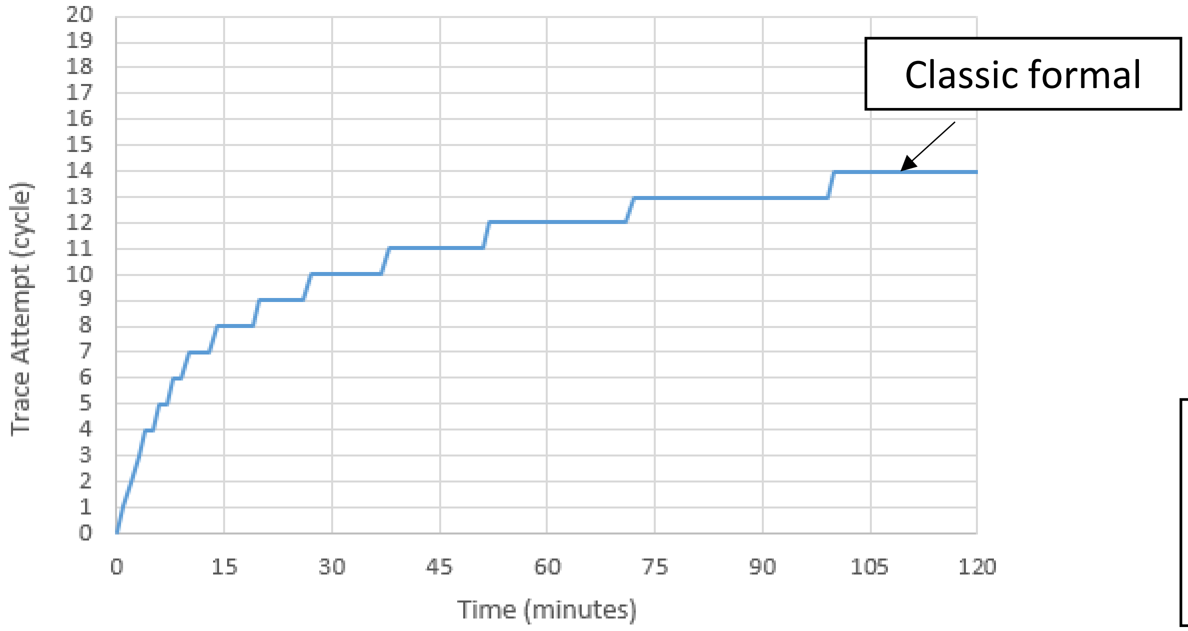
partially analyze areas

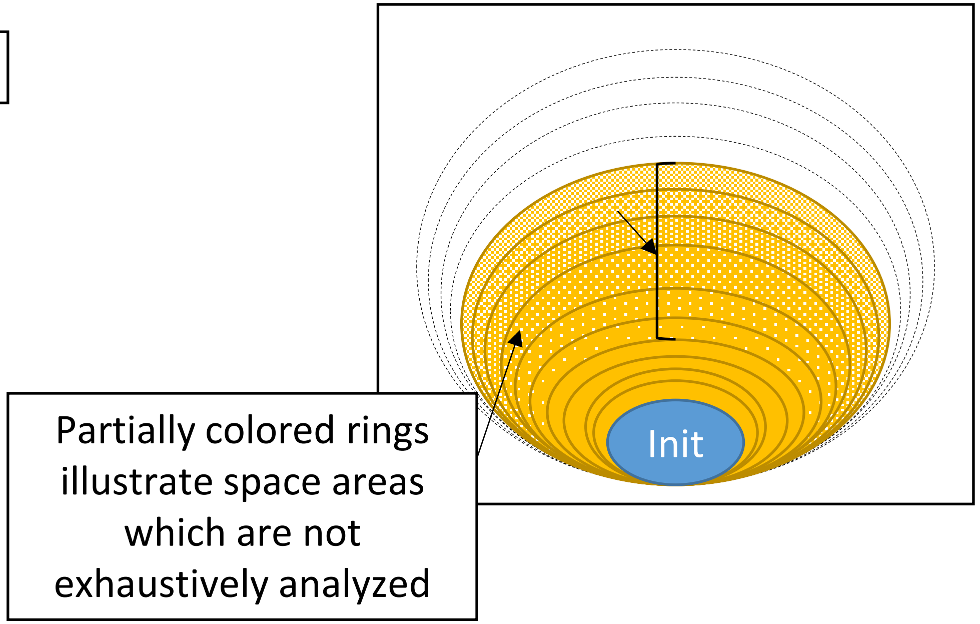


cadence

Use Case 1: Stretch Proof Bound

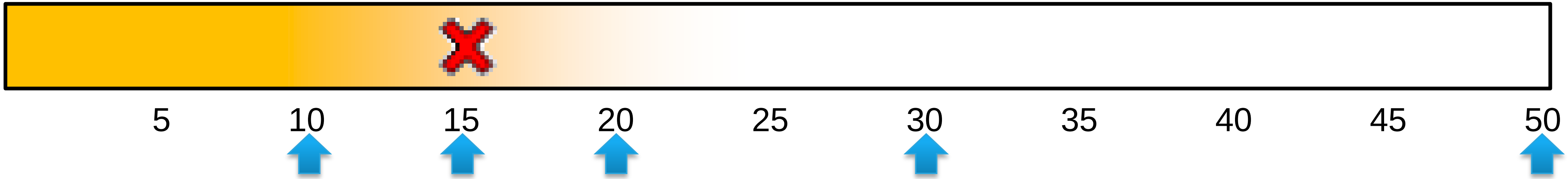
- **Partial State Space Analysis**: Limited formal search
  - Avoids search getting stuck on specific state area by partial analysis





Use Case 2: Y Swarm

Example



	Type	Name	Engine	Bound
	Assert	AST_M_overflow	B	24 -
	Assert	AST_M_rcmd_overflow	B	24 -
	Assert	AST_M_wcmd_overflow	B	23 -
	Assert	AST_M_wcmd_stable	B	19 -
	Assert	AST_M_rcmd_flags	B	16 -
	Assert	AST_M_wcmd_flags	B	13 -
	Assert	AST_M_wcmd_flags_valid	B	8 -
	Assert	AST_M_rcmd_flags_valid	B	8 -
	Assert	AST_S_integrity	B	8 -

Y Swarm

	Type	Name	Engine	Bound	Time
	Assert	AST_M_overflow	B	24 -	7.5
	Assert	AST_M_rcmd_overflow	B	24 -	9.5
	Assert	AST_M_wcmd_overflow	B	23 -	9.3
	Assert	AST_M_wcmd_stable	B	19 -	8.0
	Assert	AST_M_rcmd_flags	B	16 -	9.3
	Assert	AST_M_wcmd_flags	B	13 -	8.3
	Assert	AST_M_wcmd_flags_valid	B	8 -	6.6
	Assert	AST_M_rcmd_flags_valid	B	8	3.4
	Assert	AST_S_integrity	B	8 - 15	4.2

Classic formal reaches a bound of 8 cycles

Quickly find failure using Y Swarm

Agenda

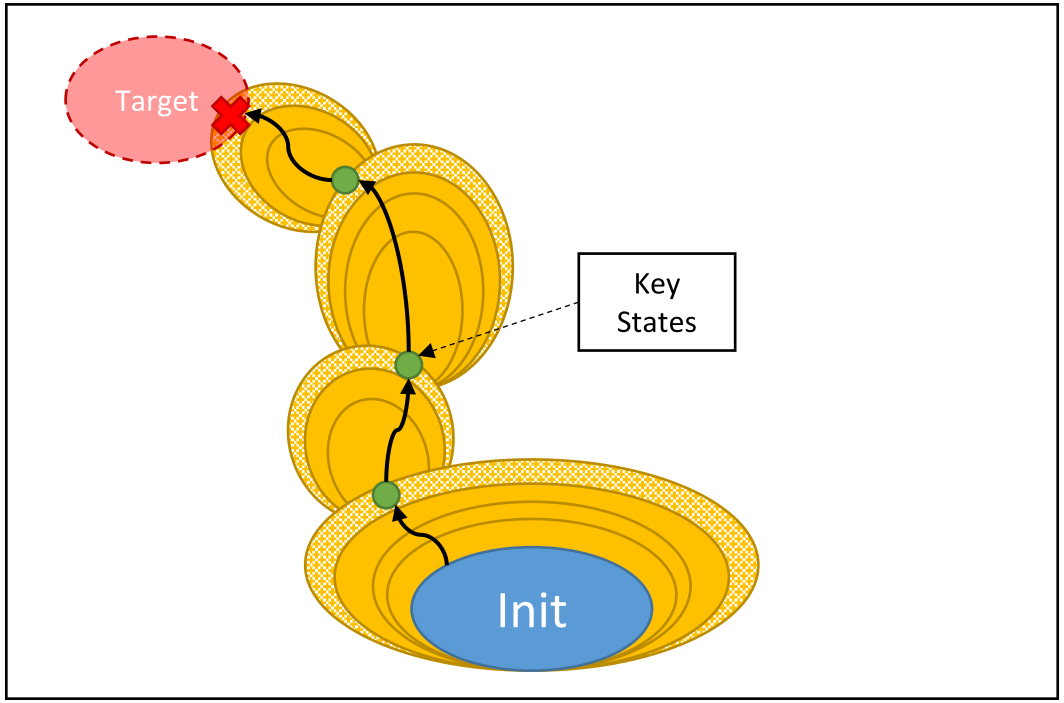
- Introduction to Bug-Hunting
- Y Swarm
- **Z Swarm**
- Guidepointing

Z Swarm
Automatically guide formal search using key states

Formal	Bug-Hunting (Semi-Formal)	Simulation

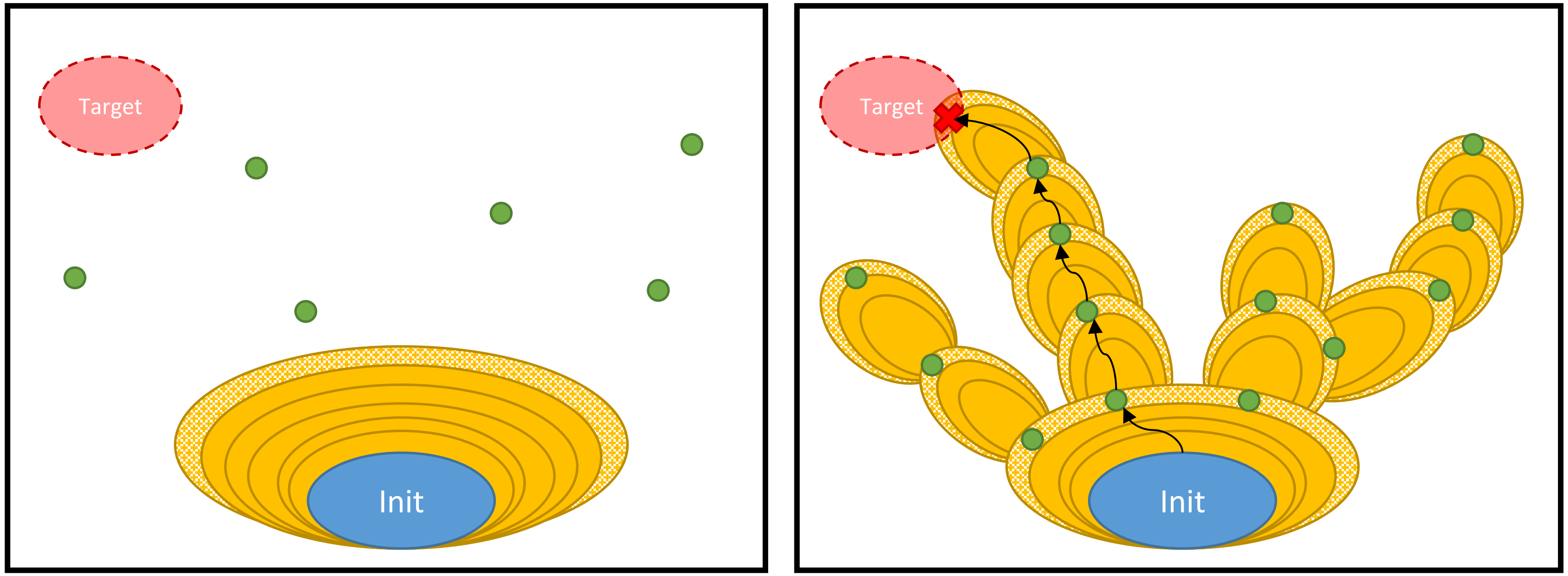
Z Swarm

- Use key states to guide formal analysis
- Effective use requires specification of relevant **Key States**
  - Interesting states of the design, directing the tool to search specific state space areas
- Enable massive parallel search
  - Customers utilize 10s-1000s parallel jobs in single session



Key States

- Key states have to be:
  - **Close** to each other, so we can go from one state to the next
  - **Diverse**, so we can hit the same scenario in multiple ways



Not effective: Key states are far apart
No advantage over Classic Formal

Very Effective: We can jump through key states
Hit much deeper scenarios than Classic Formal

cadence

Case study 1 – directing key states toward bugs

- Customer silicon bug:
  - "FSM stuck in state FIFO_FULL"
  - "token counter continues to increment"
  - `ast_BUG: assert property ( fsm==ST_FULL ##1 fsm==ST_FULL |-> $stable(token_cnt) )`

- Results:
  - Try 0: Wrote property and tried to run with BMC ➔ bound of 253 in 24hrs
  - Try 1: Added key states for Z Swarm ➔ fsm==FULL in 48hrs (8067 cycles)
  - Try 2: Further improved and directed key states ➔ fsm==FULL in 2hrs (8k+ cycles)

- Observations:
  - BMC/full proof engines were not effective for target design/depth
  - Generic key states can help Z Swarm to reach very deep targets
  - Directing them towards desired scenario is even more effective

Case Study 2

Options for Resolving the Problem

- Mutate the DUT
  - For example: downsize RAMs and FIFOs
- Over-constrain the DUT's input stimulus
  - Limit how DUT inputs are driven to a subse
- Divide and conquer DUT
  - Chop DUT into pieces
  - Prove properties about each piece separat
- Initial value abstraction
  - Reset design into a mature state (effective

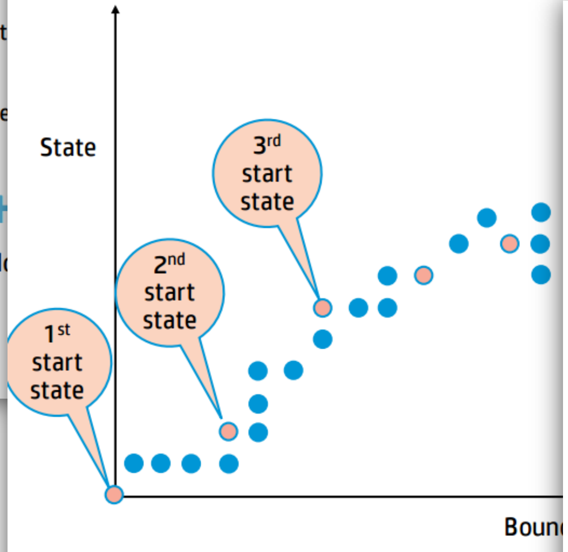
  OR

Go Bug H

[Let the machines d

4

Strategy: Advance into Deep State Space by Successively Jumping to New Start States



Hewl

8

Results

	Testcase1	Testcase2
Testcase Name	"Datapath Bridge"	"Request Router"
Duration	4 weeks	6 weeks
Total Number of Bugs Found	25	22
Number of "Deep State-Space" Bugs Found	6	1
Longest trace illustrating DUT bug	415 clock cycles	150 clock cycles
Extra effort for "Bug Hunting in Deep State-Space"	**5 days** (See Note 1)	**1 day**

NOTE 1: includes methodology development

13

Hewlett-Packard JUG-2015

Agenda

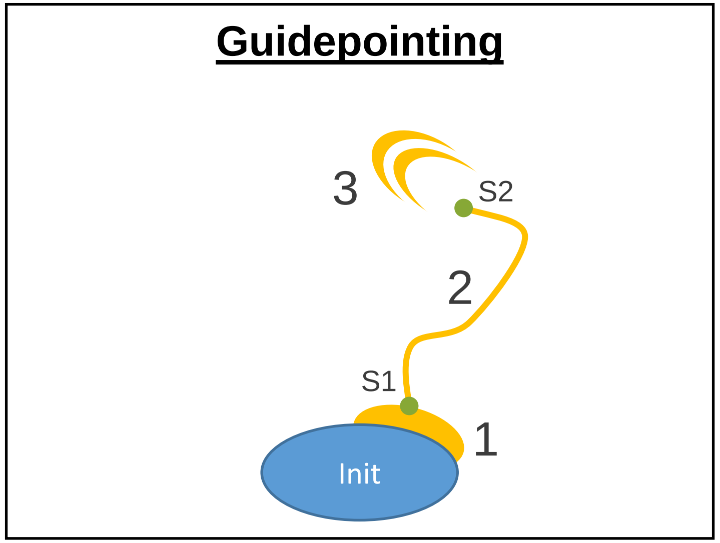
- Introduction to Bug-Hunting
- Y Swarm
- Z Swarm
- Guidepointing

Guidepointing
Manually guide analysis using formal and simulation

Formal	Bug-Hunting (Semi-Formal)	Simulation

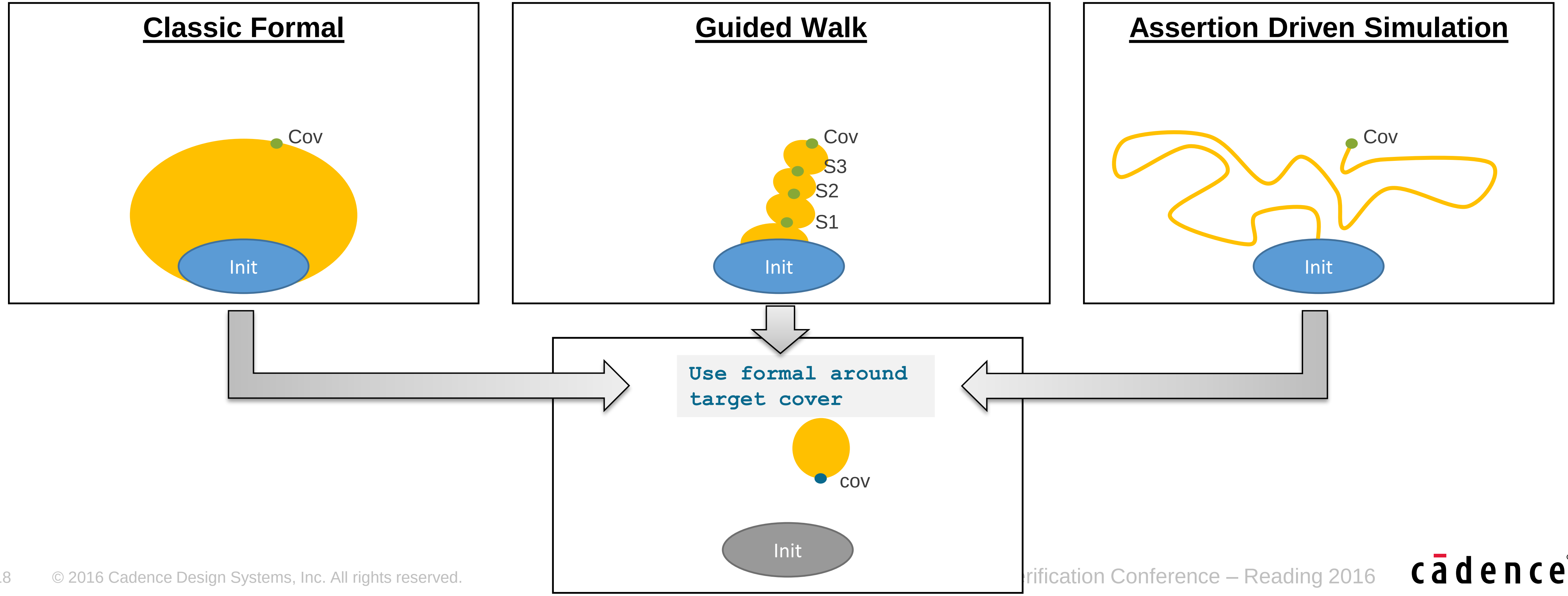
Guidepointing

- Use to tunnel through interesting states using different engines/strategies
  - Similar to Z Swarm, but in a more controlled way



Guidepointing

- Typical to want to verify assertions after hitting a certain scenario
- Several ways to hit scenario: Classic Formal, Z Swarm, Simulation



cadence

cadence®