



Vennsa OnPoint™

Beyond Debug

Confidential and proprietary
Vennsa Technologies, Inc. © 2011

Vennsa Technologies

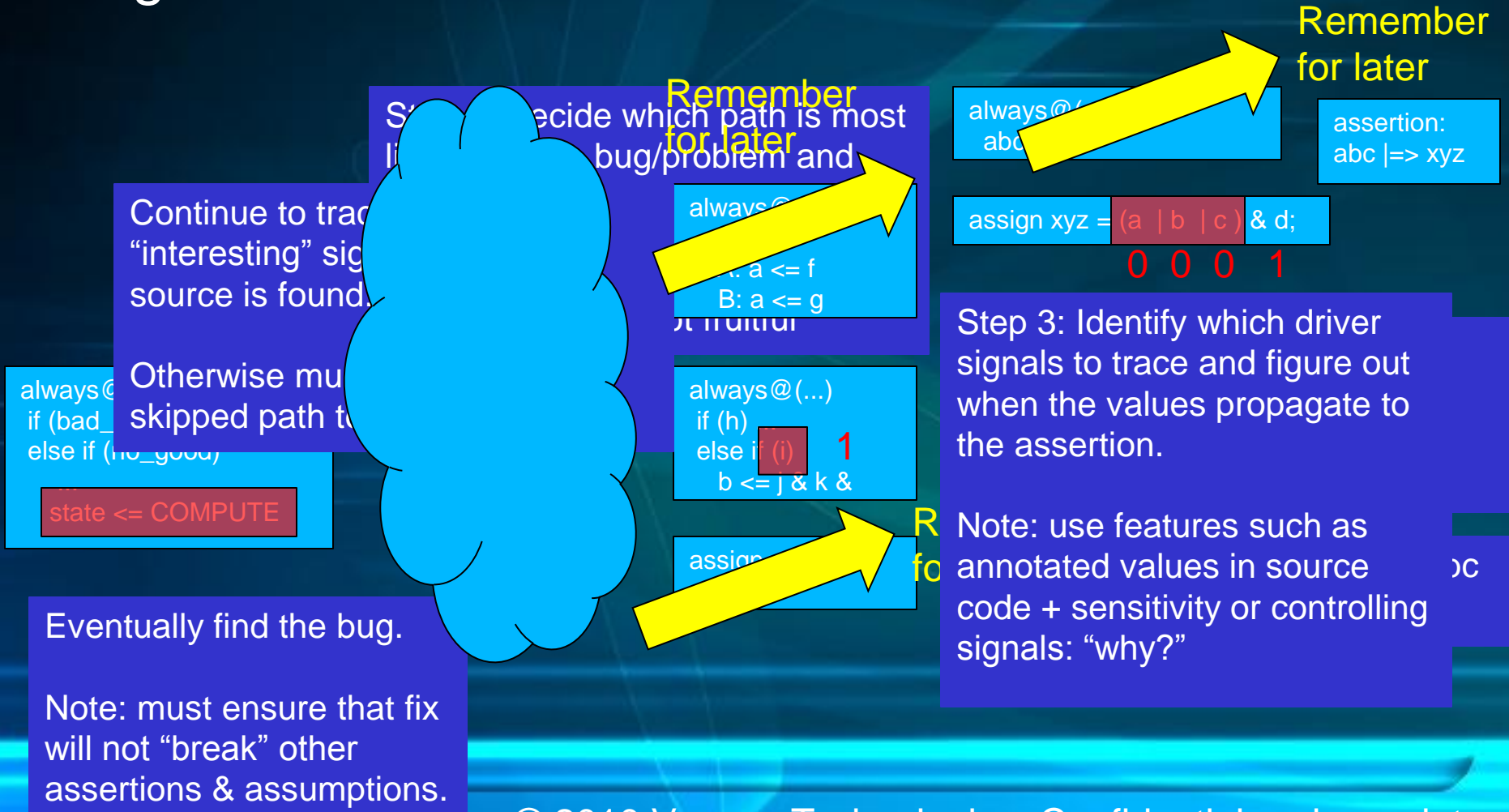
- First EDA Company Dedicated to Debug Automation
 - Spin-off from University of Toronto (2004), Incorporated 2006
 - World leaders in debugging
 - 15+ years research, 50+ publications (IEEE, ACM)
 - 5 pending patents
- Funding
 - Funded by private investors and special investments from the governments of Ontario and Canada (OCE, NRC, SRED)
- Team
 - Management: Dr. Veneris, Dr. Safarpour, Lavi Lev (ex Cadence VP)
 - Advisors: Experienced EDA and semiconductor executives
 - Sales and Support: EDA veterans in US and Japan: 50+ years
 - Technical: 10+ engineers

Debug Without OnPoint

- How is debugging done today?
 - Trace signals through waveform viewers and source code viewers
 - Navigation, exploration tools
- Debugging tools and environments help
 - Verdi, Debussy : dedicated debug/navigation tools
 - Questa, Incisive, DVE : have built-in debug features
 - JasperGold, 0-in, Magellan, IFV : helpful debug features for formal
- OnPoint is a drastically different breed of tool. Let's see how...

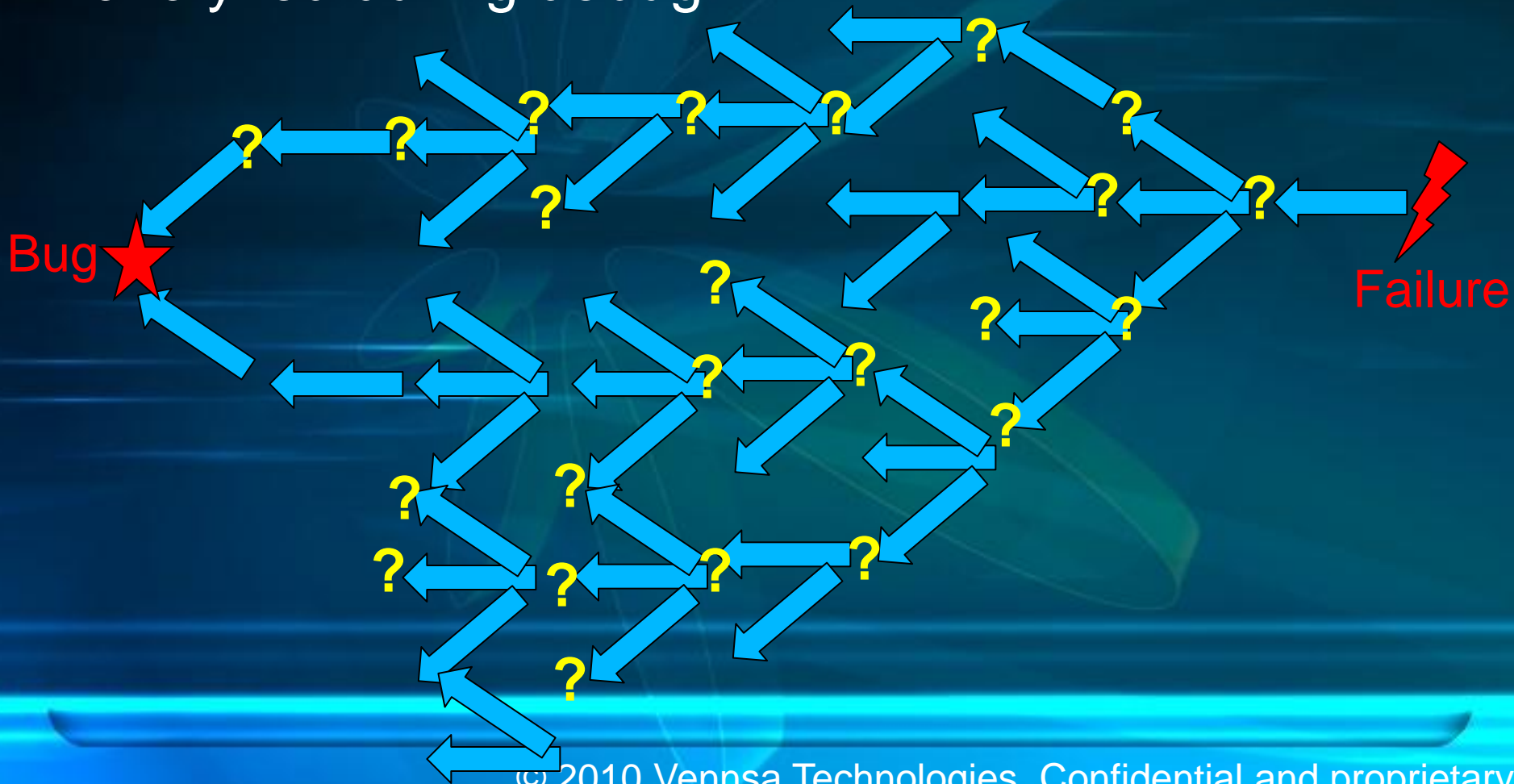
Debug Without OnPoint

- With traditional debugging you need to trace signals based on values:



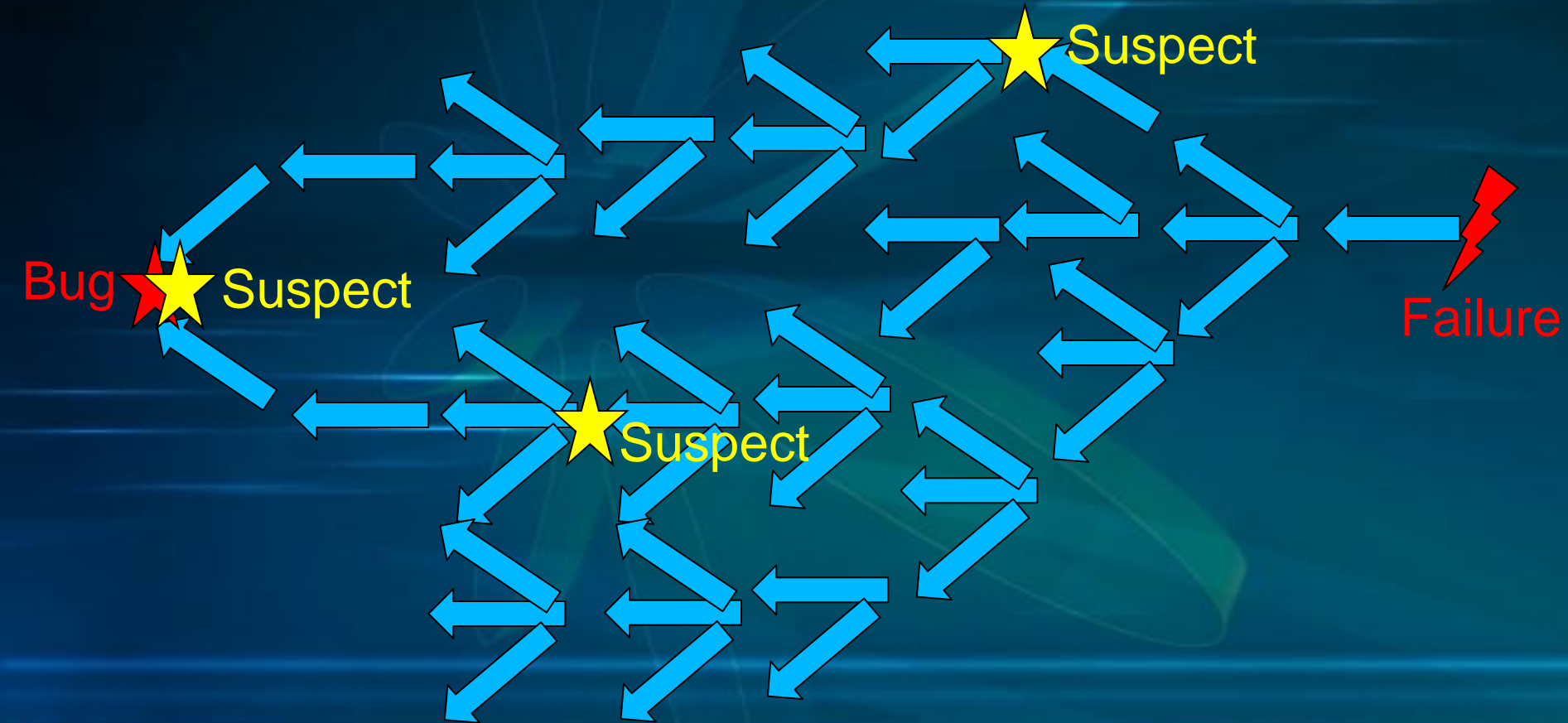
Debug Without OnPoint

- In other words, a tree of source code must be analyzed during debug



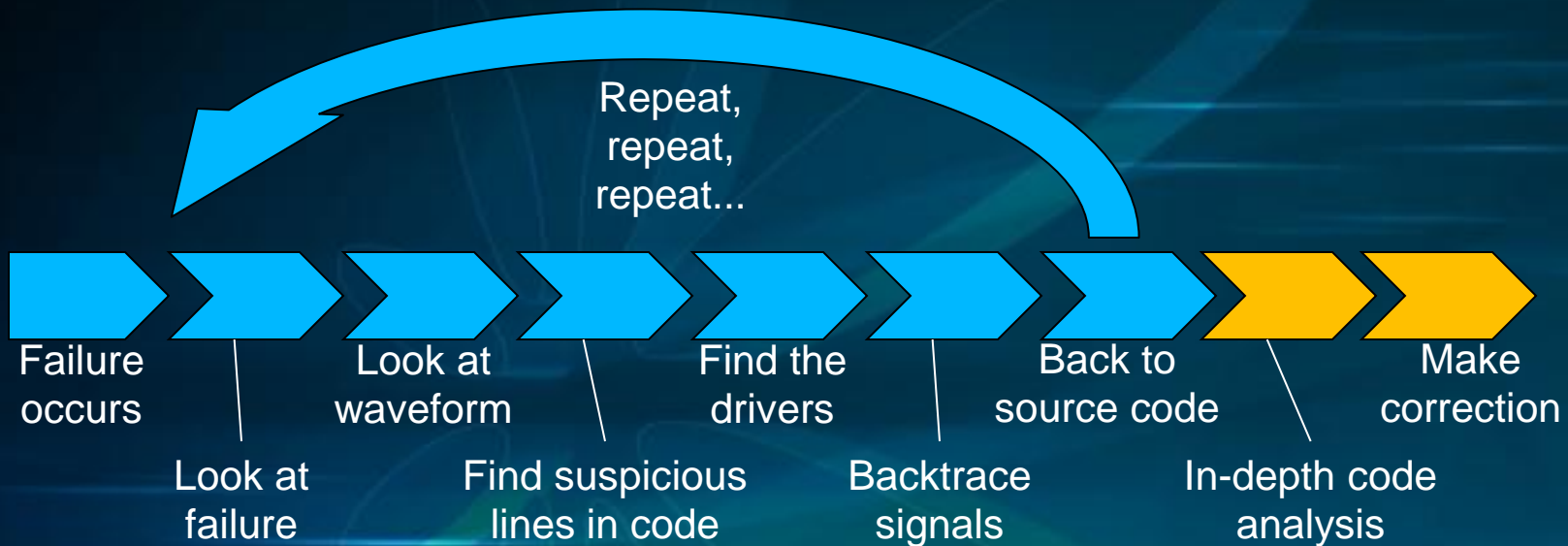
Debug With OnPoint

- OnPoint does the analysis and identifies which RTL lines of code can fix the problem without any tracing



Debug Pain: Root cause analysis

- Root cause analysis is manual and time consuming



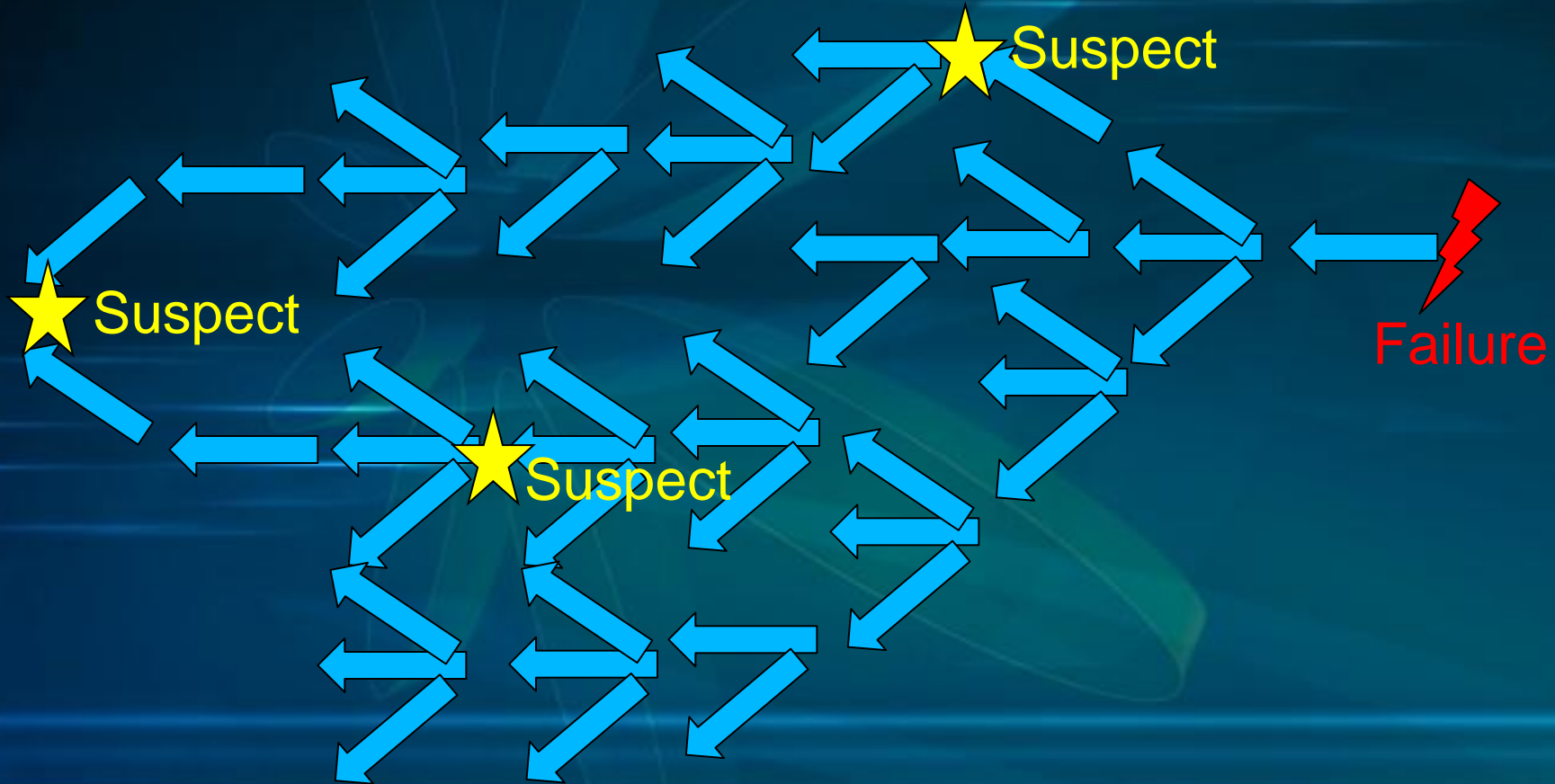
- OnPoint automates most of the tedious debugging tasks



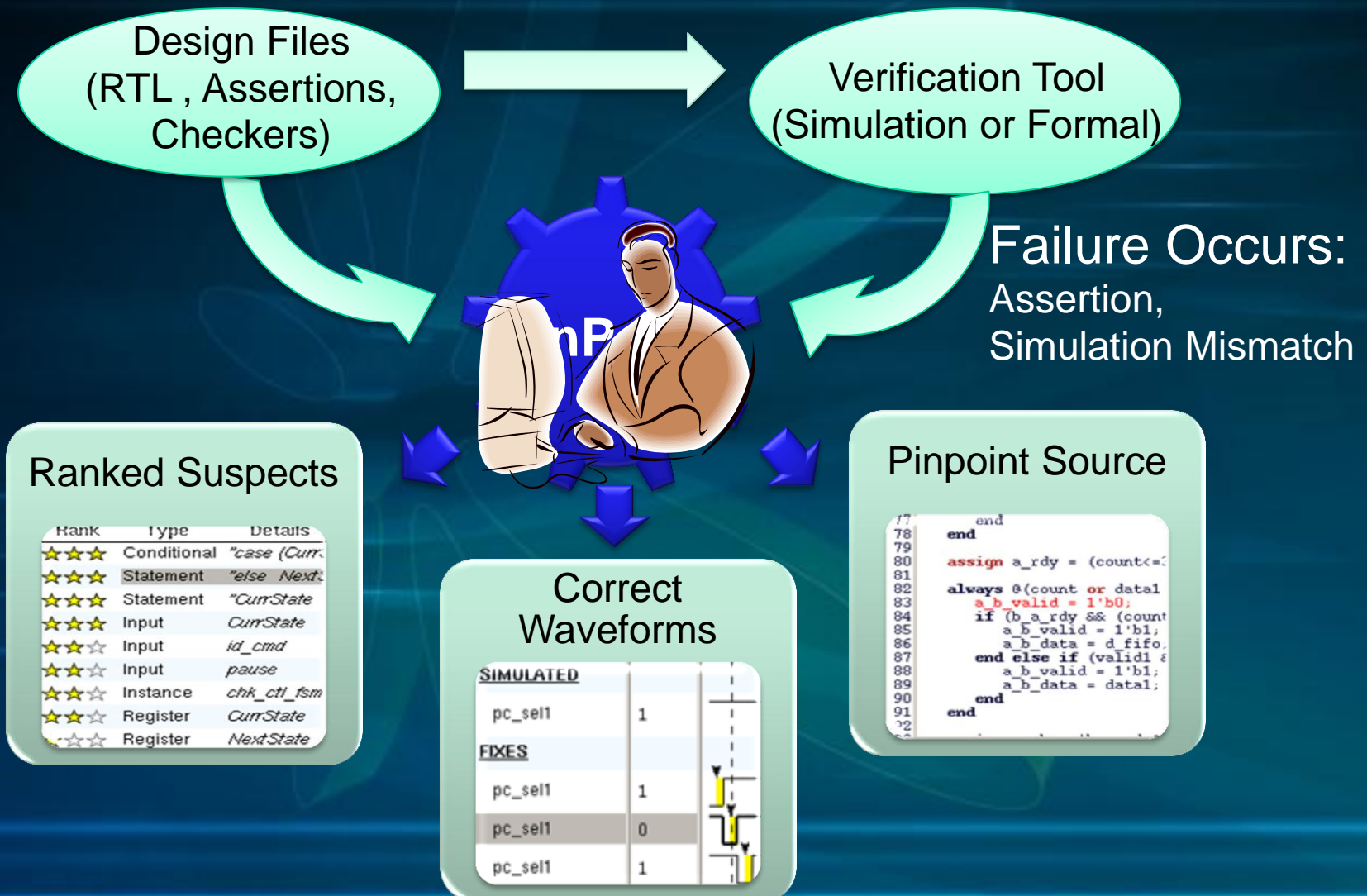
**Save hours/days per bug
Weeks/Months in design**

Hidden Problem in debug

- OnPoint output all candidate bugs as suspect
- User can see all and judge which is the bug to be fixed



Vennsa OnPoint



Vennsa *OnPoint*

- OnPoint Diagnoses every failure automatically
 - *Suspects* are returned to user
- Suspects provide *insight* into failures
 - Providing powerful *Signatures*
 - More information than error messages
- Suspects are used for root cause analysis
 - RTL constructs: *statements, expressions, signals, etc.*
 - Locations where design can change to fix bug
 - Suspects include *time* and *fix* value

Example: OnPoint suspects

The screenshot displays the Vennsa OnPoint interface with three main sections:

- Top Section:** A circuit diagram of a pipeline. A blue arrow points to a specific block in the diagram, labeled "Suspect with connectivity information".
- Middle Section:** A source code window showing Verilog code. A blue arrow points to a line of code, labeled "Suspect in source". The code includes an `assign` statement and an `always` block.
- Bottom Section:** A "Suspects" panel. On the left, a "Suspect Tree" lists various components. A blue arrow points to a specific suspect, labeled "Ranked Suspects". On the right, "Suspect Details" are shown, including a "Suspect Fix Hint" and "Suspect Times". A blue arrow points to the "Suspect Times" field, labeled "Suspect time information".

Annotations and labels are overlaid on the image:

- Suspect with connectivity information** (points to the circuit diagram)
- Suspect in source** (points to the source code)
- Suspect fix hint** (points to the suspect details)
- Suspect time information** (points to the suspect times)
- Ranked Suspects** (points to the suspect tree)

Advantage of OnPoint

- Reduce total debug time
 - 30%~50% reduction of debugging time
- Enable to fix the bug which should be fixed actually

Start Filter Function

- Find the most start point of root cause suspect
 - Can check forward, not backward
 - top down analysis --- debug effectively
- Good for system-level debugging
 - reduce debugging process
 - easy to find the bug which should be fixed actually in the system level view

Start Filter Function

Vennsa OnPoint - ethernet-fsdb.vennsawork

File View Diagnose Tools Results Help View mode: Limited Schematic

Tree Search Filelist

Modules Instances

- ▷ ALWAYS_133 i441
- ▷ ALWAYS_134 i442
- ▷ ALWAYS_135 i443
- ▷ eth_maccontrol maccontrol1
- ▷ eth_macstatus macstatus1
- ▷ eth_miim miim1
- ▼ eth_registers ethreg1
 - ▷ Ports (370)
 - ▷ Nets (119)
 - ▷ Primitives
 - ▷ ✓ ALWAYS_30 i707
 - ▷ ALWAYS_31 i708
 - ▷ ALWAYS_32 i709
 - ▷ ALWAYS_33 i710
 - ▷ ALWAYS_34 i711
 - ▷ ALWAYS_35 i712

Source Cone Schematic Cone/Source

Sig Inst ethreg1

Console Suspects

Target Assertion: Target Debug Instance:

View mode: tree flat 0+ Stars Sync tree

★★★★ Statement "temp_wb_dat_o_reg <=#Tp temp_wb_dat_o"

★★★★ Statement "assign temp_wb_dat_o = (((RegCs) & ~w)"

★★★★ Statement "assign MAC_ADDR0_Wr = Write & MAC"

Show hidden suspects

Details Hints Waveform

File: rtl/eth_registers.v

Line(s): 373

Module: eth_registers

Name: eth_registers/assign_25_MAC_ADDR0_Wr

Type: Statement

Sort Suspects By: rank

Suspect Filter: Manual

- instances
- signals

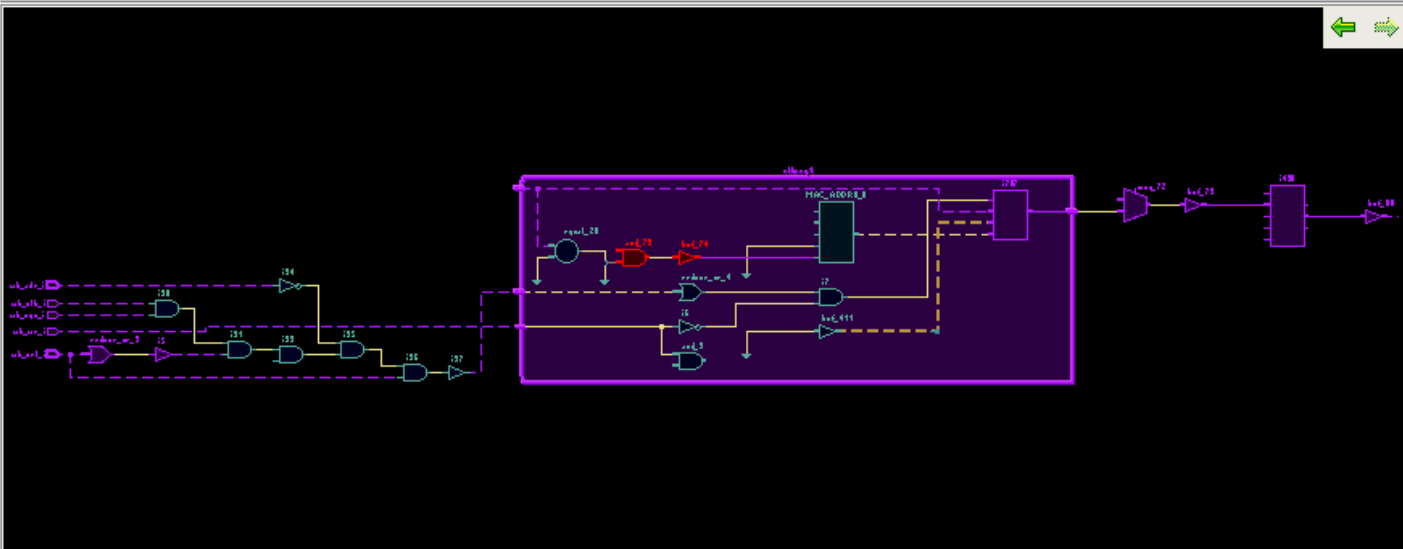
Start Filter Function

Vennsa OnPoint - ethernet-fsdb.vennsawork

File View Diagnose Tools Results Help View mode: Limited Schematic

Tree Search Filelist Source Cone Schematic Cone/Source Sig Inst ethreg1

Modules	Instances
▷ ALWAYS_133	i441
▷ ALWAYS_134	i442
▷ ALWAYS_135	i443
▷ eth_maccontrol	maccontrol1
▷ eth_macstatus	macstatus1
▷ eth_miim	miim1
▼ eth_registers	ethreg1
▷ Ports (370)	
▷ Nets (119)	
▷ Primitives	
▷ ✓ ALWAYS_30	i707
▷ ALWAYS_31	i708
▷ ALWAYS_32	i709
▷ ALWAYS_33	i710
▷ ALWAYS_34	i711
▷ ALWAYS_35	i712



Console Suspects

Target Assertion: Target Debug Instance:

View mode: tree flat 0+ Stars Sync tree

- ☆☆☆ Statement "temp_wb_dat_o_reg <= #Tp temp_wb_dat_o_reg"
- ☆☆☆ Statement "assign temp_wb_dat_o = ((RegCs) & ~w)"
- ☆☆☆ Statement "assign MAC_ADDR0_Wr = Write & MAC"

Show hidden suspects

Details Hints Waveform

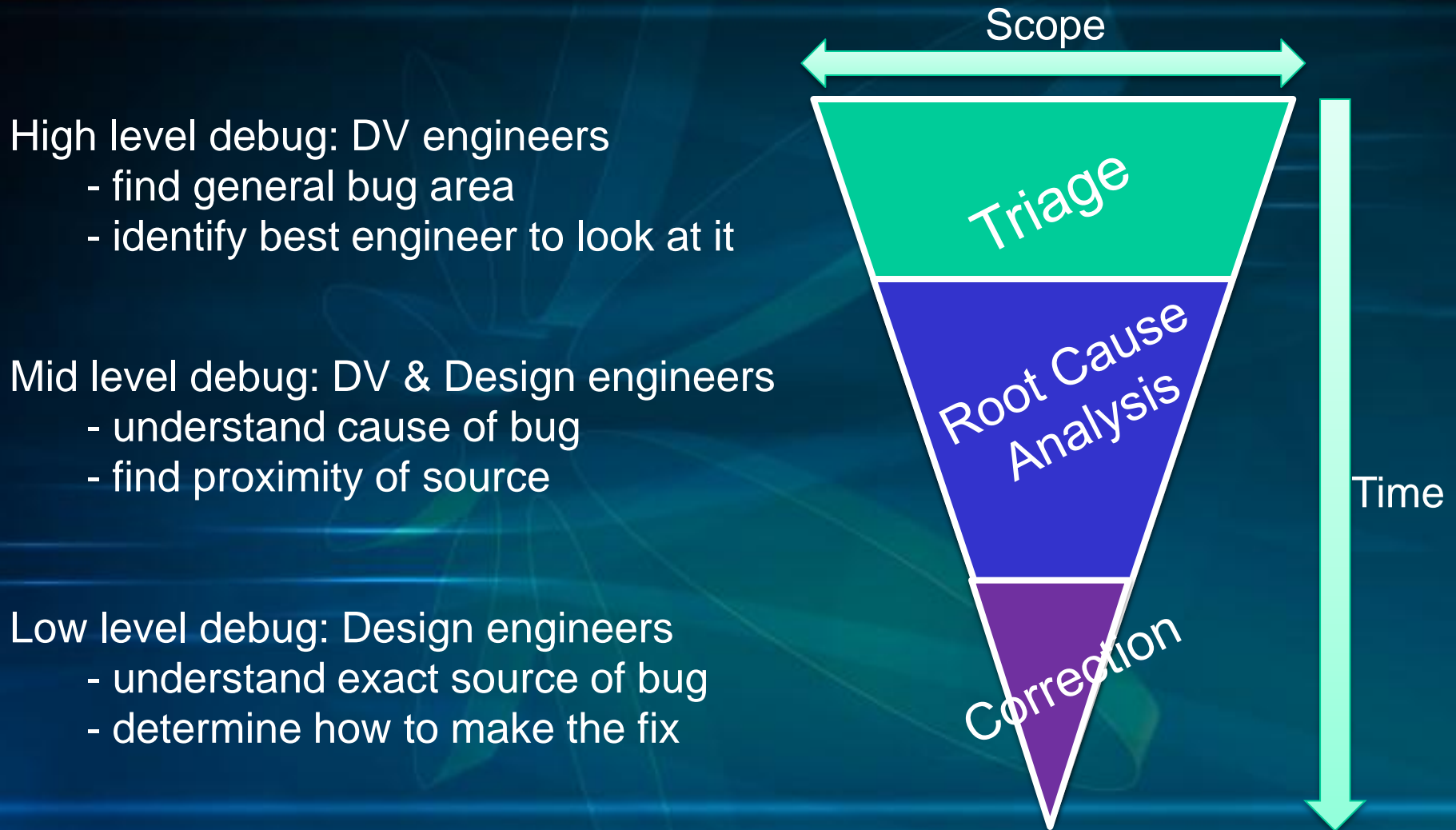
File: rtl/eth_registers.v
Line(s): 373
Module: eth_registers
Name: eth_registers/assign_25_MAC_ADDR0_Wr
Type: Statement

Sort Suspects By: rank

Suspect Filter: Manual

- ☒ instances
- ☒ signals

Debug Scopes



Debug Pain: Triage

Error: checker CHK42 failed
Error: checker CHK63 failed
Error: checker CHK42 failed
Error: assertion A23 failed
Error: monitor Mon1 failed
Error: assertion A24 failed
Error: checker CHK42 failed
...



- Which ones are related, which are not?
- Same failures/same reasons? How many?
- Which ones to “file” as a bug? To who?
- What’s the source: design, testbench, env?

Nightly
Regression tests

Probably
yours



Not my problem

Not mine

Yours

Yours

Designer: Joe

Designer: Tim

DV: Bob

**Takes time &
Wastes time**

Binning example

- Two different bug sources, one error point



- Two different error points, same bug source



Triage with OnPoint

Error: checker CHK42 failed
Error: checker CHK63 failed
Error: checker CHK42 failed
Error: assertion A23 failed
Error: monitor Mon1 failed
Error: assertion A24 failed
Error: checker CHK42 failed
...

Nightly
Regression tests



Signatures



Bin1



Bin 2



Bin N



Designer: Joe



Designer: Tim

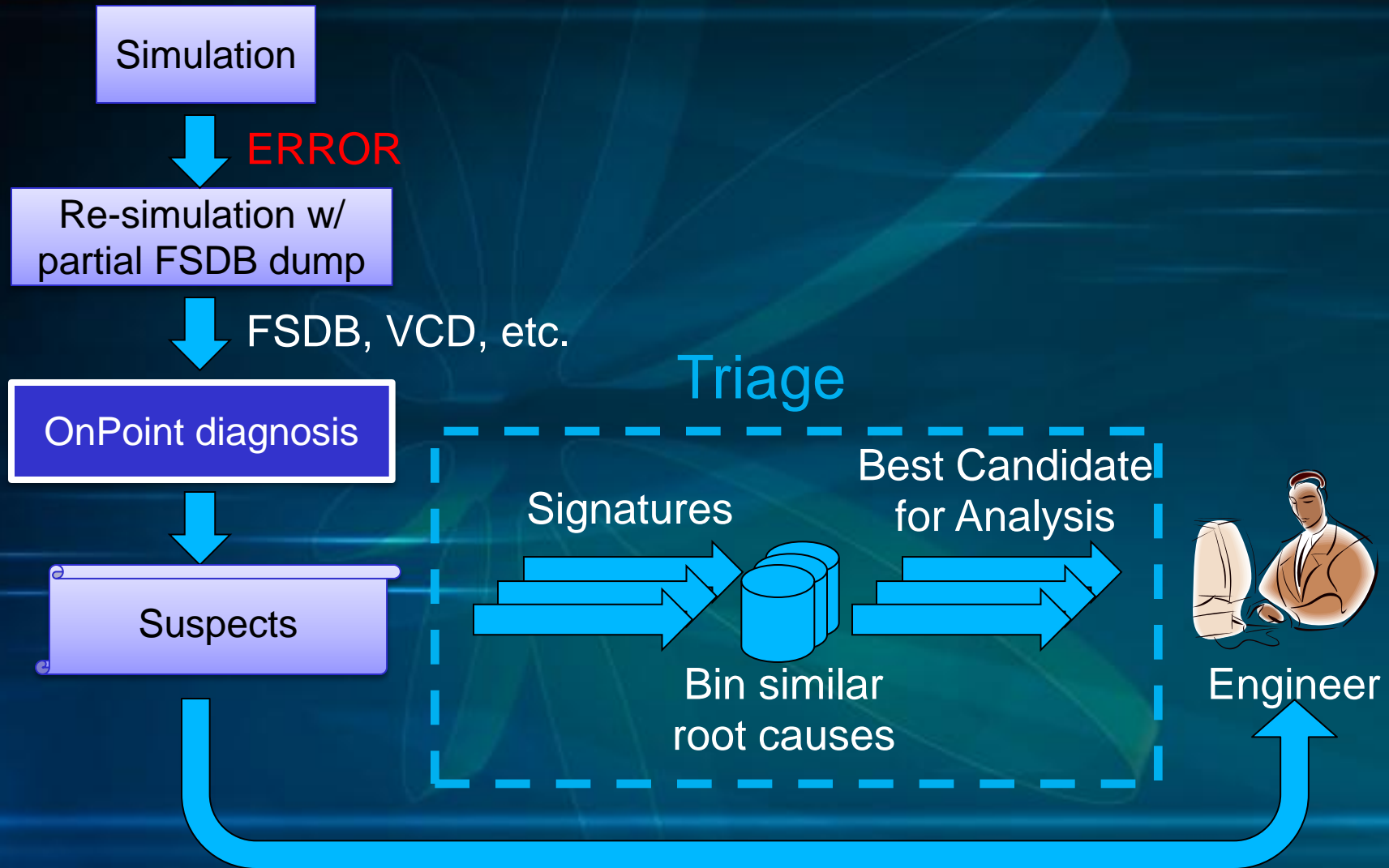


DV: Bob

Binning based on

- RTL problem
 - modules, registers, RTL lines, etc.
 - Similar sources (irrespective of failure point)
- Testbench checker problem
- Testbench stimulus problem

Verification and Debug Flow



OnPoint Applications

OnPoint accelerates debug in the following application domains.

RTL Debug

Root cause analysis of
RTL designs

Assertion Debug

Root cause analysis of
assertion failures

Failure Triage

Binning of failures
based on root cause



X Propagation

Find source of X in RTL
or netlist designs

Netlist Debug

Root cause analysis of
gate level netlists

Formal Verification

Debug of formal
counter examples

Support Plan

- Integration with current debugger tool
 - Verdi --- supported
 - other debugger tools
- I/F with system level language
 - Bluespec --- supported soon
 - other system level language developed by EDA vendors
- Integration with IP
 - design IP, verification IP, checker
- Integration with specific verification environment
 - integration with scoreboard
- Subjects in Testbench

Integration with Verdi

The image displays the Verdi EDA tool interface, which is used for verifying digital designs. The interface is divided into several panes:

- Top Left:** Verilog code editor showing a module named `risc_core`. The code defines a prescaler and a watchdog timer. A line of code is highlighted: `prescaler_out <= #1 prescaler_out_next & prescaler_out_r1 & ~prescaler_clr;`.
- Top Center:** A logic diagram showing the internal structure of the prescaler, with inputs `3'd0` through `3'd7` and outputs `3'd0` through `3'd7`.
- Top Right:** A second Verilog code editor showing the same module, with a different line highlighted: `prescaler_out <= #1 prescaler_out_next & prescaler_out_r1 & ~prescaler_clr;`.
- Bottom Left:** A "Suspect Tree" pane showing a list of suspects. The top suspect is `prescaler_wdt (0) - prescaler_wdt_v`. Below it are several statements and registers.
- Bottom Center:** A waveform viewer showing a clock signal `clk` and several output signals: `prescaler[7:0]`, `prescaler_out`, `prescaler_out_next`, and `prescaler_clr`. The time scale is 100ns.
- Bottom Right:** A "Suspect Tree" pane showing a list of suspects. The top suspect is `prescaler_wdt (0) - prescaler_wdt_v`. Below it are several statements and registers.

I/F with Bluespec

The screenshot displays the Vennsa OnPoint IDE interface, showing a Bluespec design and its associated signals and waveforms.

Source Code (mkDMA.v):

```
1562 // we finish a read when we see the correct respInfo on the mmu
1563 // response fifo.
1564 assign CAN_FIRE_RL_finishRead_2 =
1565     responseDataFs_2SRDY_enq &&
1566     (mmuRespF_taggedReg[44] || mmuRespF_rw_enq$whas) &&
1567     x_h3487 == 10'd2 ;
1568 assign WILL_FIRE_RL_finishRead_2 =
1569     CAN_FIRE_RL_finishRead_2 && !WILL_FIRE_RL_startWrite_2 ;
1570 // rule RL_finishWrite_3
1571 // This rule waits for the write to finish
```

Signal List:

Type	Signal	Value
wire	..._FIRE_RL_finishWrite	=1
wire	..._FIRE_RL_finishRead_1	=1
wire	...ponseDataFs_1SEN_enq	=0
wire	...RL_markTransferDone	=0
wire	...ponseDataFs_1EN_enq	=0
wire	..._FIRE_RL_finishWrite	=1
wire	...onseDataFs_1f_wPtr	=0
wire	...LL_FIRE_RL_startRead	=0
wire	...mmuReqF_rw_enq\$whas	=1
wire	...eDataFs_1f_wPtr\$EN	=0
wire	...AN_FIRE_RL_startRead	=1
wire	...onseDataFs_1SRDY_enq	=1
wire	..._FIRE_RL_finishRead_1	=0
wire	...RL_mmuRespF_rule_deq	=1
wire	..._FIRE_RL_startRead_3	=0
wire	mkDMA.dmaEnabledRs_0SEN	=0
wire	...muRespF_taggedReg\$EN	=1
wire	...RL_mmuRespF_rule_deq	=1

Waveform: The waveform shows the timing of various signals over time, with a marker at 702.00 s.

Suspects: The suspect tree shows the design hierarchy, with the selected suspect being `CAN_FIRE_RL_finishRead_1` in the `mkDMA (1) - mkDMA.v` module.

Details: The details pane shows the signal `CAN_FIRE_RL_finishRead_1` with a value of 1, and the waveform shows the signal's timing over time.

Vennsa Technologies

Thank you

For more information and evaluation license contact:

info@vennsa.com

America
San Jose, CA
408-400-3708

Headquarters
Toronto, ON
416-829-0091

Example: OnPoint suspects

The screenshot displays the Vennsa OnPoint interface with three main panels:

- Top Panel (Circuit Diagram):** Shows a hardware circuit diagram. A blue arrow points to a specific component with the text "Suspect with connectivity information".
- Middle Panel (Source Code):** Displays the Verilog source code for `b_a_valid`. A blue arrow points to the line `a_b_valid = 1'b0;` with the text "Suspect in source". Another blue arrow points to the `end else if` block with the text "Suspect fix hint".
- Bottom Panel (Suspects):** Contains a "Suspect Tree" on the left and "Suspect Details" on the right. The tree lists suspects like `mb`, `ma`, `always_2`, and `always_1`. A blue arrow points from the "Ranked Suspects" text to the tree. The details panel shows information for the selected suspect, including file path, line number, module, name, type, and suspect times (2650-2700). A blue arrow points from the "Suspect time information" text to the "Suspect Times" field. A "Suspect Fix Hint" box provides a hint: "RTL: Check for incorrect signal or expression on right-hand-side or incorrect timing of assignment."

Ranked Suspects

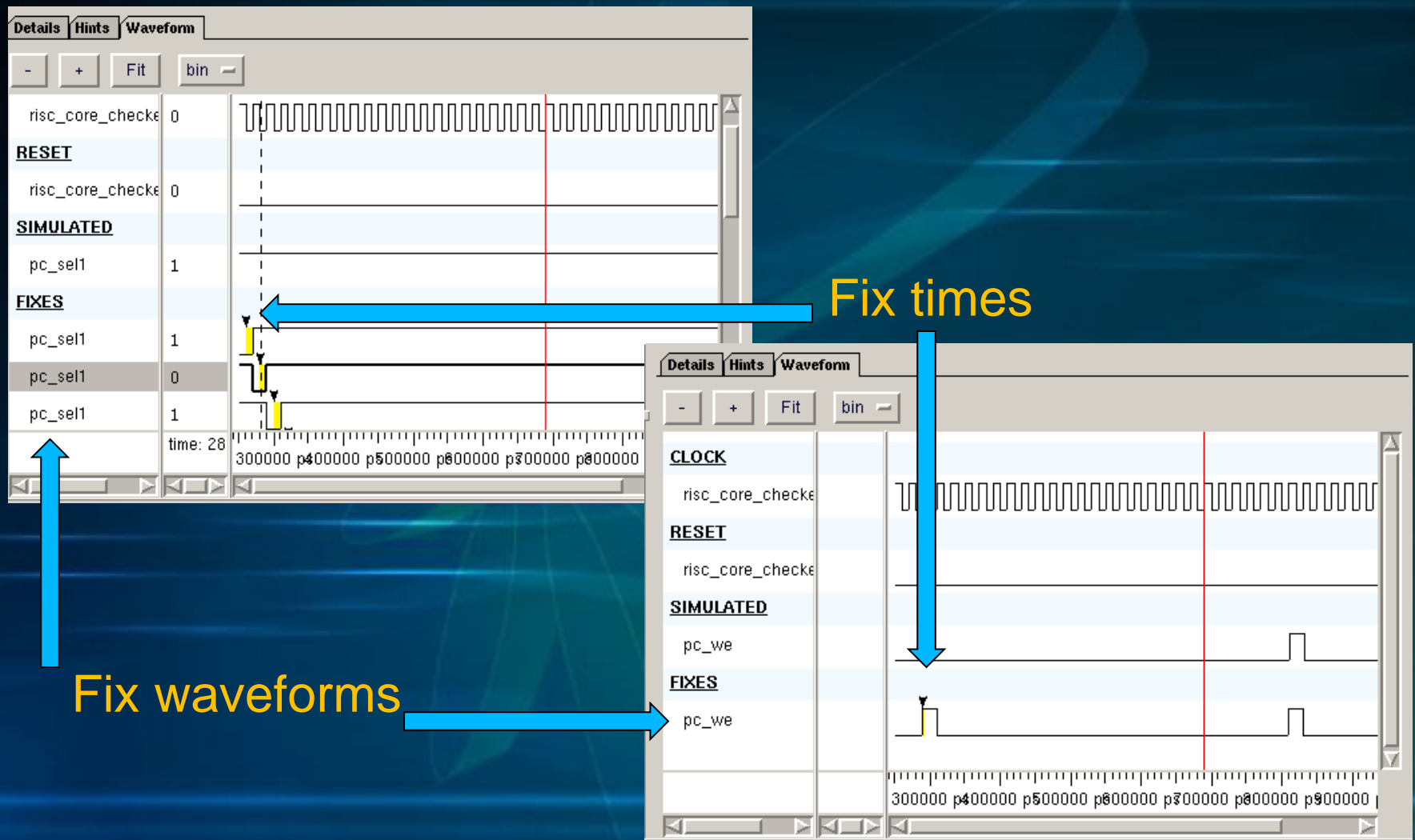
Suspect in source

Suspect fix hint

Suspect time information

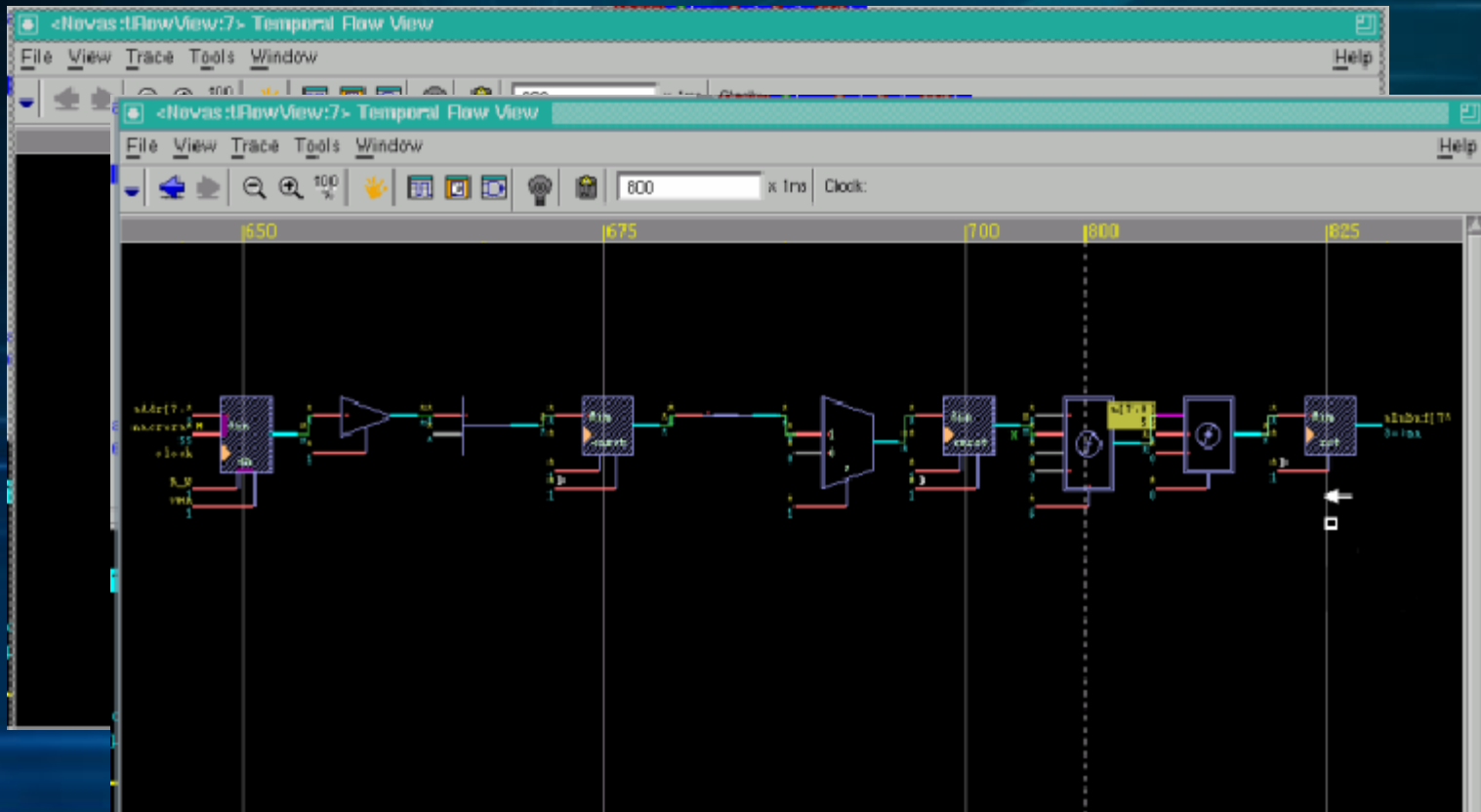
Suspect with connectivity information

Example: Fix values for suspect



Comparison with Verdi

- In Verdi there are “advanced” debug features
- Can do “Behavioral Analysis” and select “Trace this Value”
- Will show in the temporal view where the value is assigned



Comparison with Verdi

- Verdi simply traces the value back as far as it can
 - Based on value propagation
- Verdi cannot reason about “how to fix bug” and cannot trace differences in values and across functions
- OnPoint, in comparison, can determine the paths that can fix the failure
- For example, changing the select line of a mux to pick from another input
 - This bug would be missed by Verdi, but found with OnPoint automatically

