

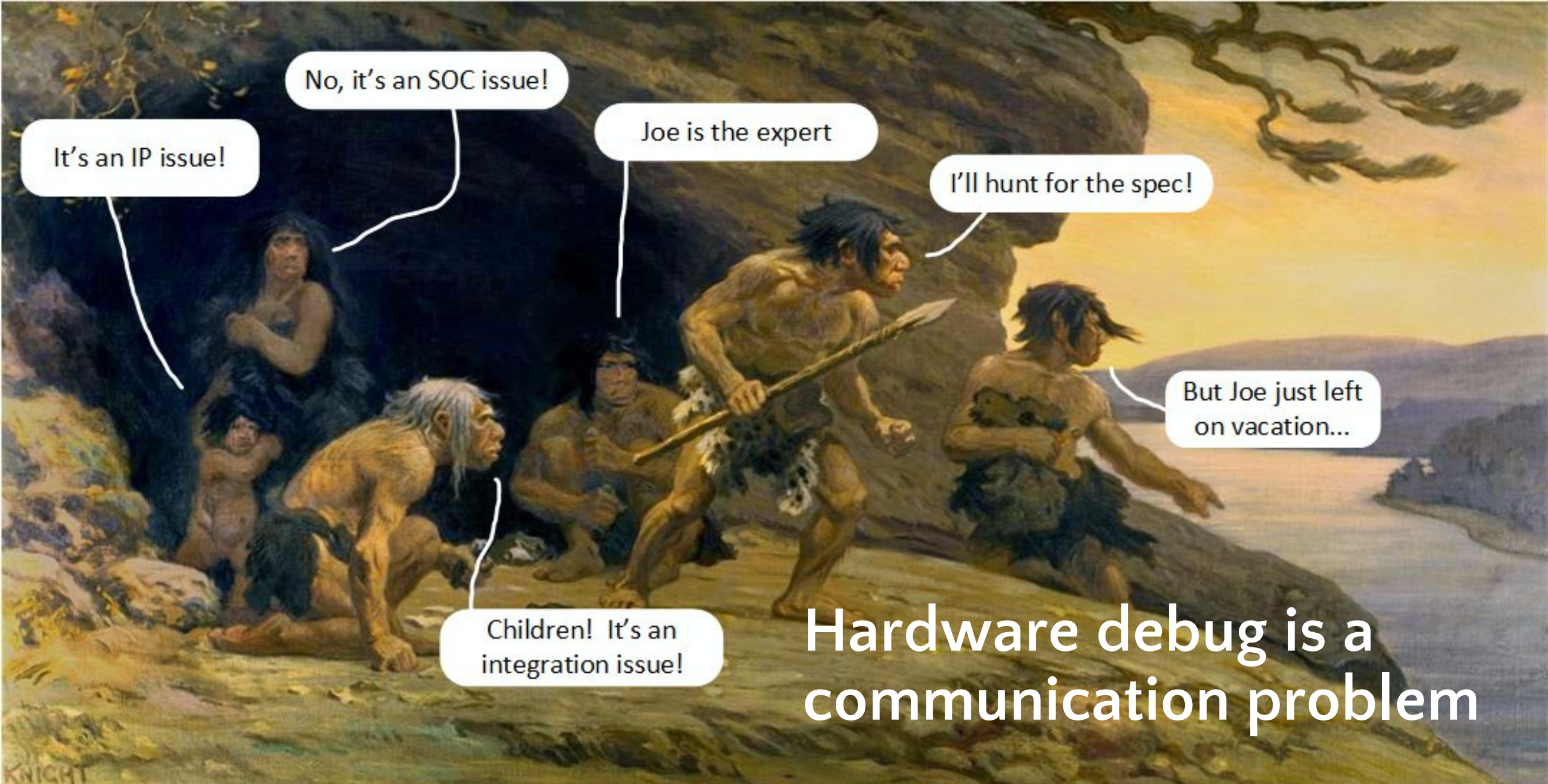


Debug Decision Trees

Erik Berg

Microsoft - Principal Engineer





No, it's an SOC issue!

It's an IP issue!

Joe is the expert

I'll hunt for the spec!

But Joe just left
on vacation...

Children! It's an
integration issue!

Hardware debug is a communication problem

Successful Debug

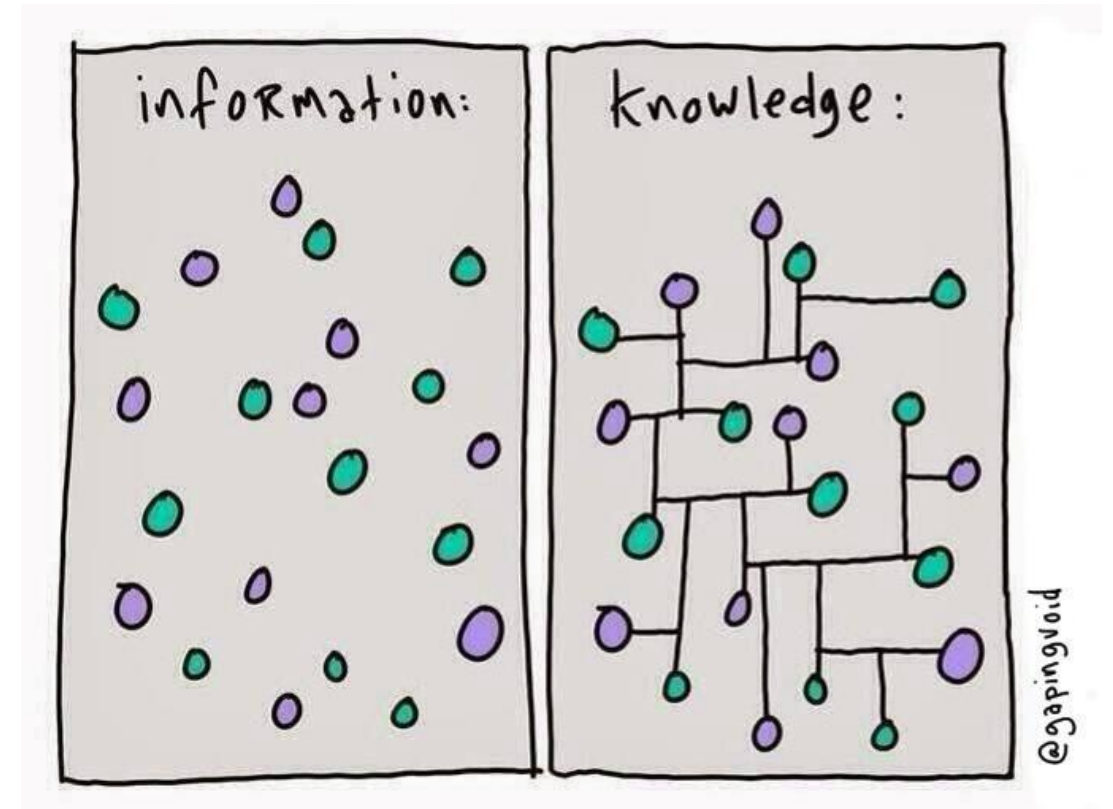
“The minimum sequence of observable events to deterministically define the path from the error message to the root cause.”

Information: Observable events

- Testbench inform statements
- Waveform events
- Transaction log files

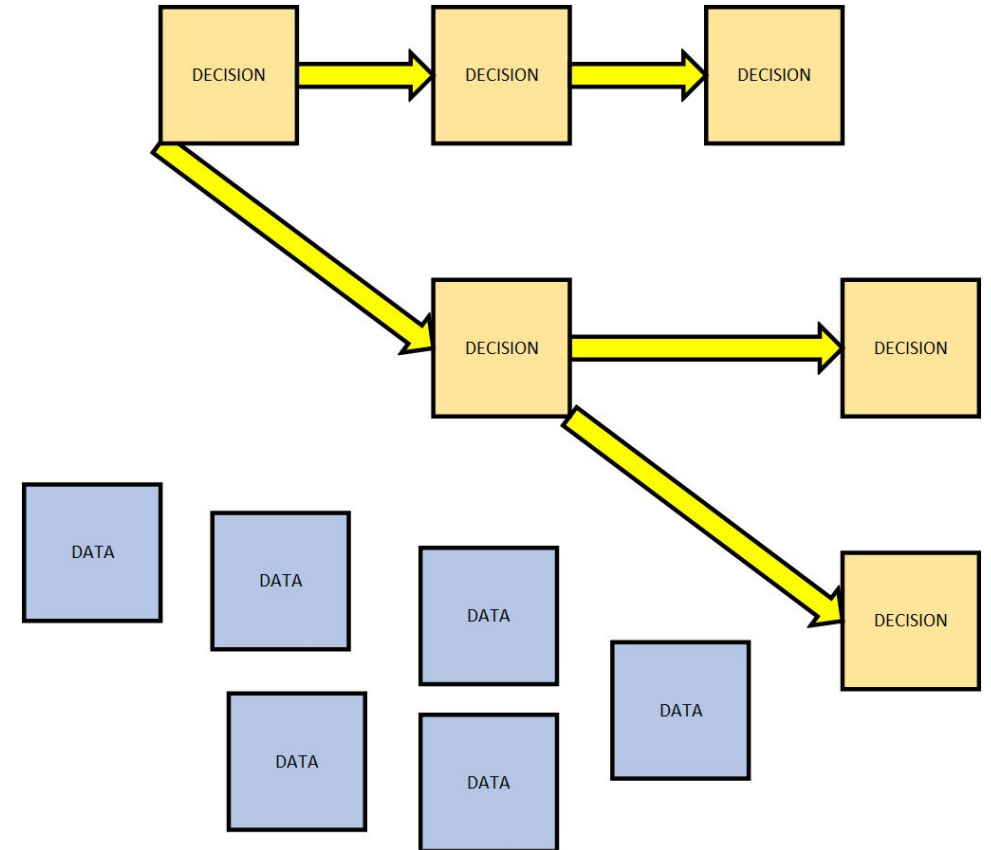
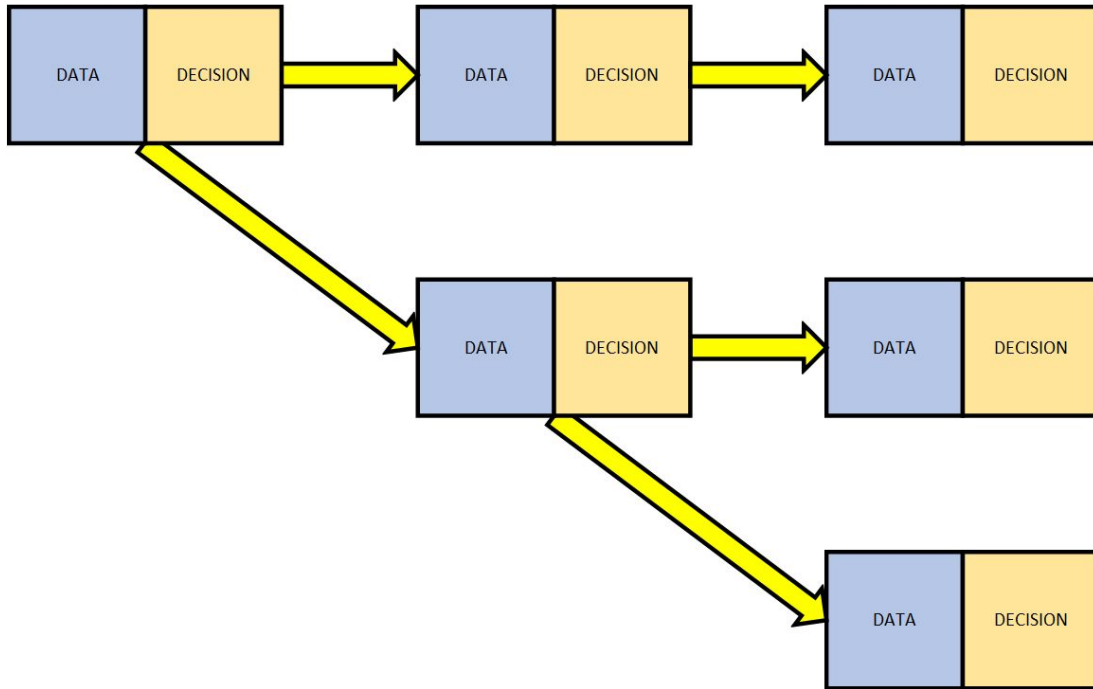
Knowledge: Connections between events

- Highly valuable expert knowledge
- Difficult to capture in text – context is key



ThenWhatTree

Separate data extraction from debug decision



DDT Usage Model

Joe debugs a failure and can describe the root cause.

Creates the root node with required context

Adds the branch for the failure

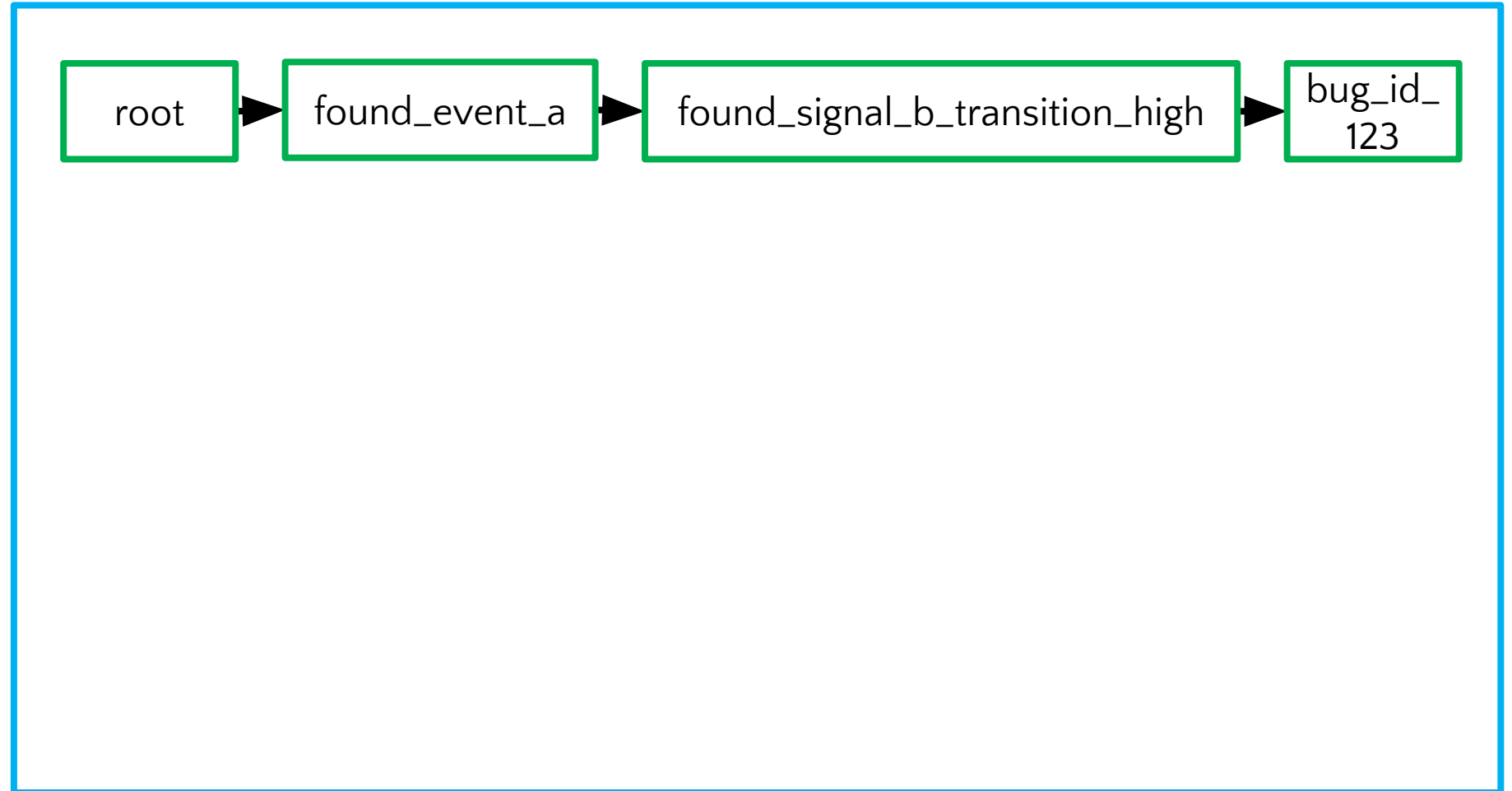
Generates .py executable



DDT Usage Model

The ddt .py executable is run as a post-processor against new failures

Match: no debug is required

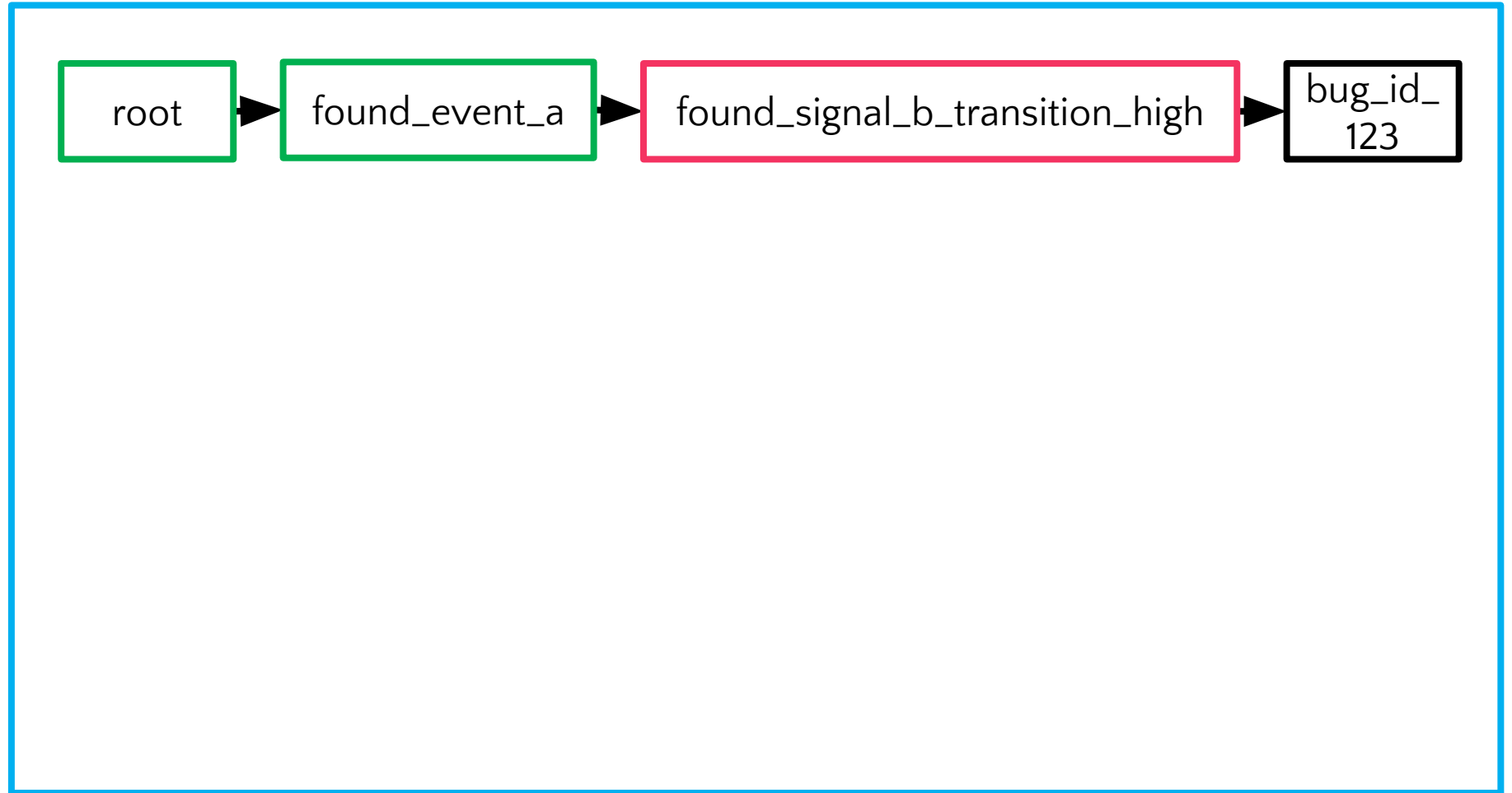


DDT Usage Model

The ddt .py executable is run as a post-processor against new failures

Match: no debug is required

No match: debug as usual



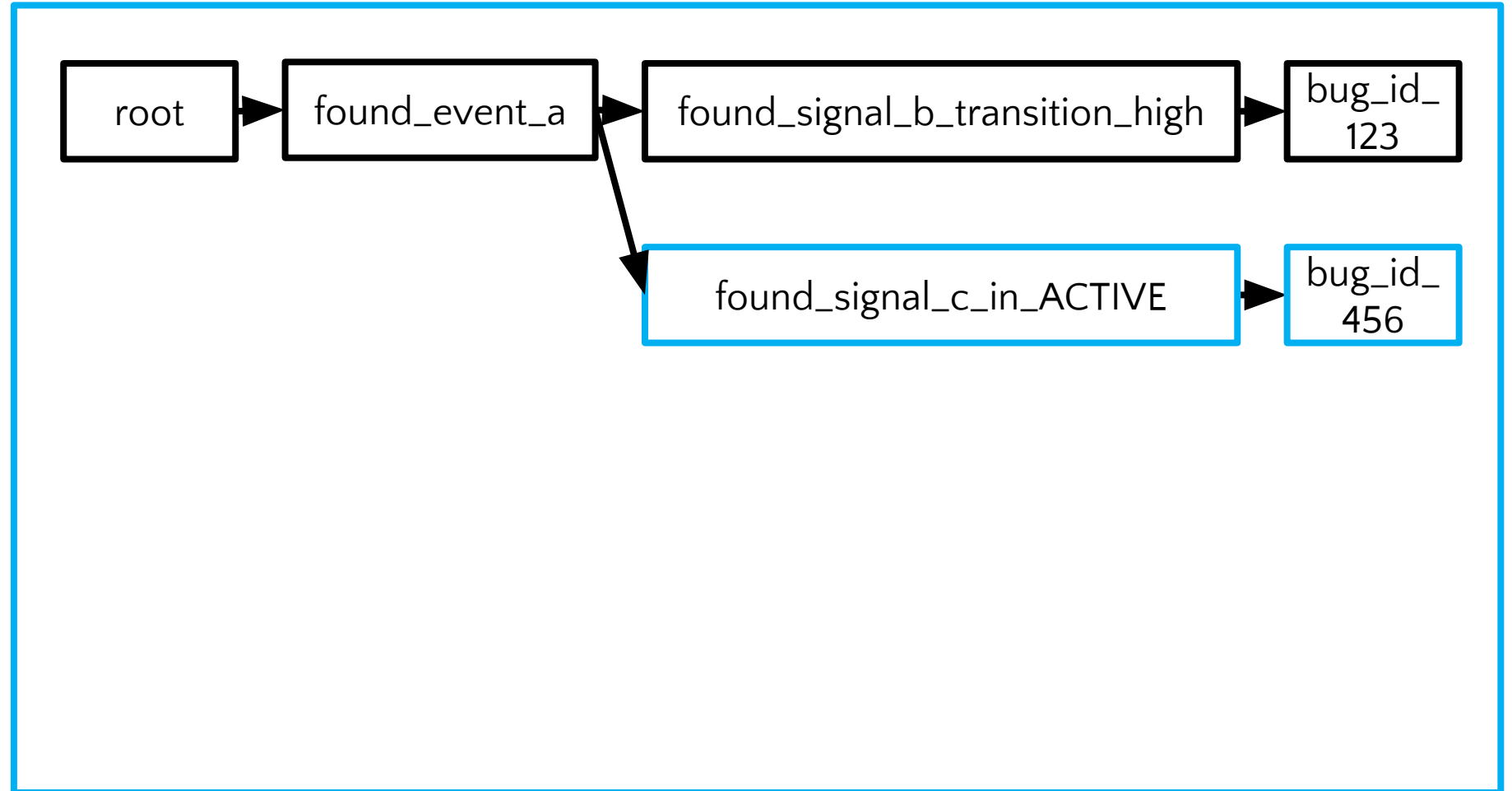
DDT Usage Model

Joe's coworker debugs a new failure

Extends the root node if new context is required

Adds the branch for the failure

Generates new .py executable

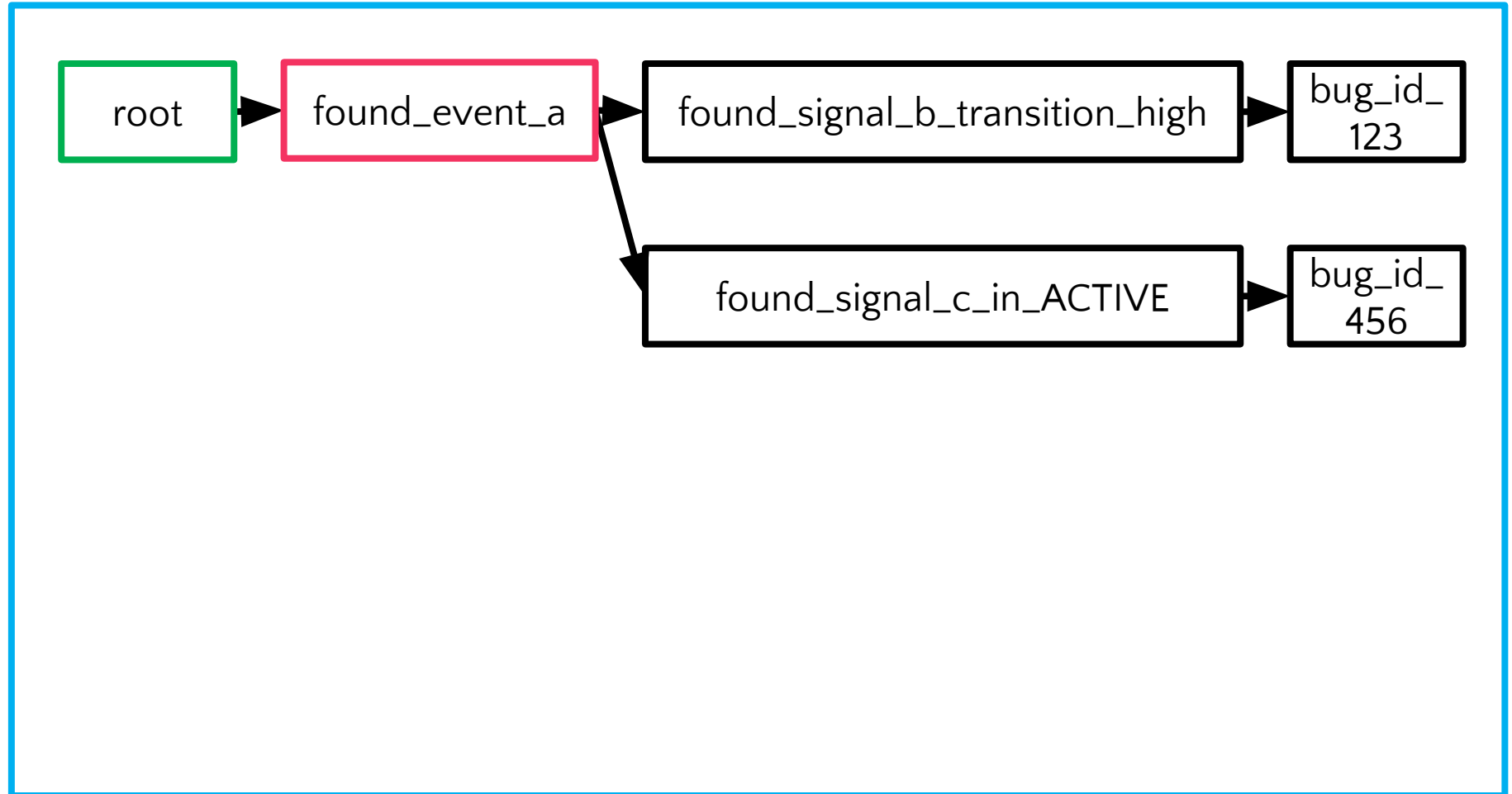


DDT Usage Model

The ddt .py executable is run as a post-processor against new failures

Match: no debug is required

No match: debug as usual



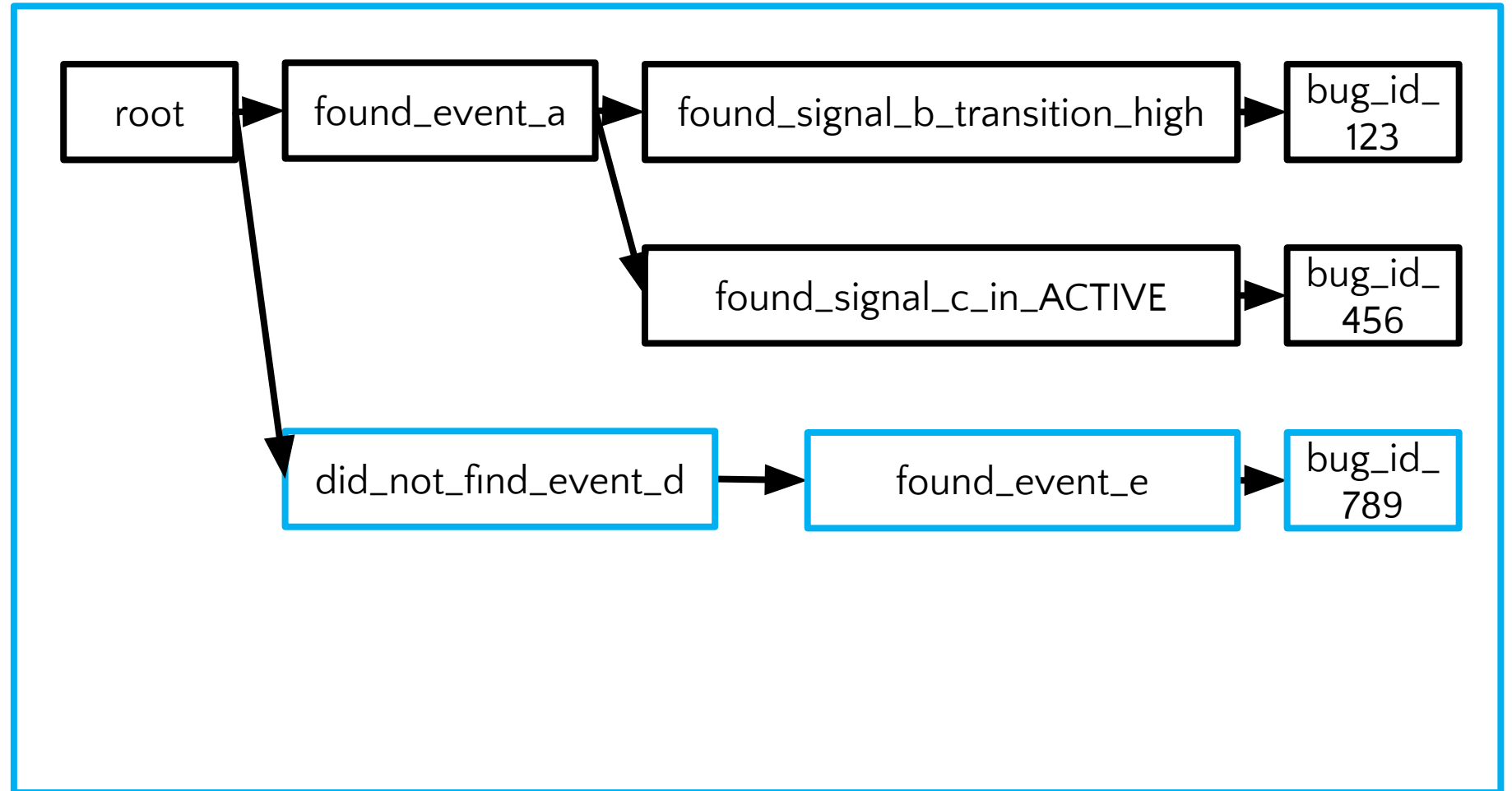
DDT Usage Model

Joe's coworker debugs a new failure

Extends the root node if new context is required

Adds the branch for the failure

Generates new .py executable



DDT GUI

Node template libraries

Decision tree structure

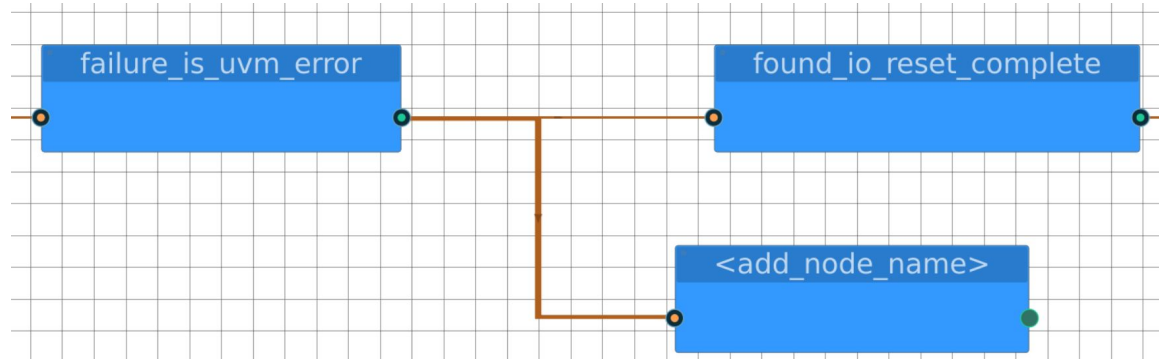
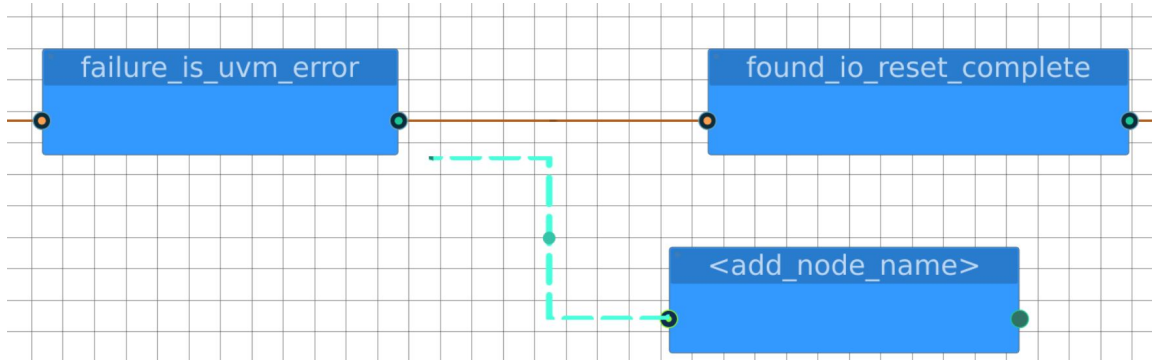
Node parameter inputs

Root node

Branch nodes

The screenshot displays the DDT GUI interface. On the left, a 'Libraries' pane lists various node templates, including 'Text-Log Libraries', 'Waveform Libraries', 'Intelligent Libraries', 'Miscellaneous Libraries', 'Customize Library', 'DummyEnum', 'DummyCommonStrings', 'DummyUtilityFunctions', 'DummyTestbenchFunctions', 'Common_Functions', 'context_check', 'context_util', 'output_message', 'Root_Node', 'base_root_node', 'Fsdb_Functions', 'found_signal_value_at_eot', 'found_signal_value_at_time', 'found_time_last_signal_value_c', 'found_time_value_next_signal_', 'found_transition_at_time', 'found_transition_to_value', 'Regexp_Functions', 'found_txn', 'Table_Functions', and 'found_entry'. The 'found_entry' node is currently selected. The main area shows a decision tree structure with a root node and several branch nodes, all represented by blue rectangles. The tree is enclosed in a dashed black box. The bottom panel, titled 'failure_is_uvm_error', shows the configuration for the selected 'context_check' node. It includes a 'Library' section with 'context_check' selected, and a 'Arguments' section with various input fields: 'Context field:' (set to 'status'), 'Context sub-field (optional):' (set to 'UVM_ERROR'), 'Check value:' (set to 'EQUAL'), 'Data type:' (set to 'STRING'), 'Range type:' (set to 'WHOLE_VALUE'), 'Range value' (set to 'NONE'), 'Tolerance type:' (set to '0'), 'Tolerance value (3% is entered as 3):' (set to '0'), 'Custom Node Output Message:' (set to 'False'), and 'Debug Switch:' (set to 'False'). At the bottom, a 'Variables List of the Data Extracted' section shows 'context: The decision tree branch elements (dictionary)'.

Connecting New Branch Nodes



Defining Branch Nodes

Querying unstructured text files:

“Search **FORWARD** from **PREVIOUS_NODE_TIME** for ‘Regular Expression 1’ ”

- Match groups are added/updated in context

The screenshot displays the configuration interface for a branch node named "failure_is_uvm_fatal". The interface is divided into several sections: "Info", "Arguments", and a list of configuration options. The "Info" section shows the "Library Name" as "found_txn". The "Arguments" section lists four regular expressions, with the first one highlighted in green. The configuration options section includes fields for "Log File Path", "Regular Expression 1", "Regular Expression 2", "Regular Expression 3", "Regular Expression 4", "Regex match type", "Search direction", "Start time", "End time", "Time delta type", "Time delta limit", "Num instances to match", "Stop search if regexp match", "Match groups that must match context", "Match groups that cannot match context", "Names of match groups to compare to 'Ignore values'", "Ignore values", "In-line match group operation", "In-line match groups to compare", "Return value if regexps not found", "Custom node output message", "Debug switch", and "Show log file line in output". The "Search direction" is set to "FORWARD" (highlighted in blue), and the "Start time" is set to "PREVIOUS_NODE_TIME" (highlighted in red). The "Regular Expression 1" field contains the regex: "(?P<error_type>UVM_FATAL)\s+.*\s@\s(?P<time>\d+)\s.*\[(?P<error_message>\S+)\].*" (highlighted in green).

Configuration Option	Value
Library Name	found_txn
Log File Path	SIM_LOG
Regular Expression 1	"(?P<error_type>UVM_FATAL)\s+.*\s@\s(?P<time>\d+)\s.*\[(?P<error_message>\S+)\].*" (highlighted in green)
Regular Expression 2	""
Regular Expression 3	""
Regular Expression 4	""
Regex match type	ANY
Search direction	FORWARD (highlighted in blue)
Start time	"PREVIOUS_NODE_TIME" (highlighted in red)
End time	None
Time delta type	FROM_START_TIME
Time delta limit	None
Num instances to match	1
Stop search if regexp match	""
Match groups that must match context	[]
Match groups that cannot match context	[]
Names of match groups to compare to 'Ignore values'	[]
Ignore values	[]
In-line match group operation	NOT_USED
In-line match groups to compare	[]
Return value if regexps not found	False
Custom node output message	""
Debug switch	False
Show log file line in output	True

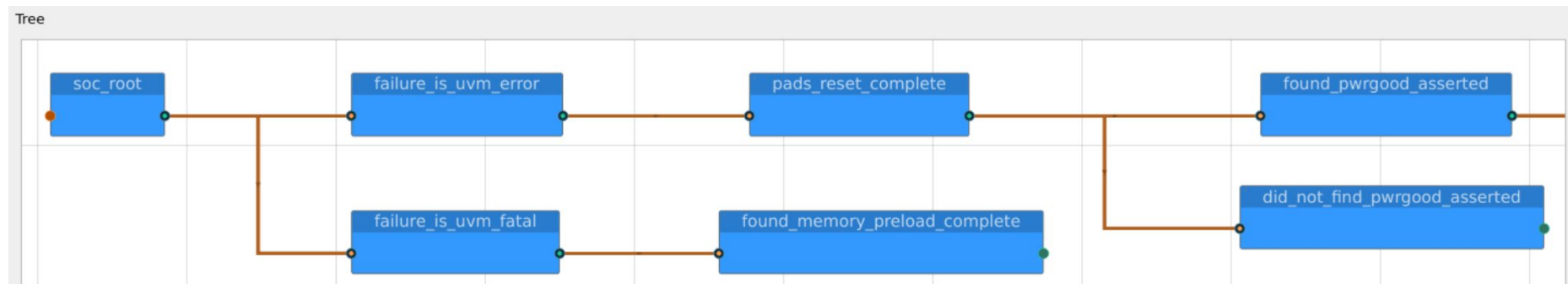


DDT Tree Output

```
soc_root: TRUE [1]
  failure_is_uvm_error: TRUE [3]
    pads_reset_complete: TRUE [22]
      found_pwrgood_asserted: TRUE [26]
        boot_fsm_did_not_reach_ACTIVE: FALSE [2]
        boot_fsm_reached_ACTIVE: TRUE [15]
          found_IROM_to_IRAM_transition: TRUE [18]
            found_ip_wakeup: FALSE [31]
            did_not_find_ip_wakeup: TRUE [32]
              did_not_find_pwrgood_asserted: FALSE [37]
            failure_is_uvm_fatal: FALSE [34]
```

Node Output:

```
[1] []
[3] ['True: Context check: "status" EQUAL "PASSED" at time 0']
[22] ['[pads_reset_complete] Found signal values at time 995.001: pads_pwrgd_i: h1, pads_reset_i: h1']
[26] ['[found_pwrgood_asserted] Found signal values at time 1195.001: pwrgd_asserted_n_o: h1']
[2] ['[boot_fsm_did_not_reach_ACTIVE] Never found signal values: porfsm_nxtstate: 0101']
[15] ['[boot_fsm_reached_ACTIVE] Found signal values at time 17555.001: porfsm_nxtstate: 0101']
[18] ['[found_IROM_to_IRAM_transition] Found signal values at time 31052.68109: program_counter: h00010000']
[31] []
[32] ['[did_not_find_ip_wakeup] Never found signal values after time 31052.68109: ip_wakeup_i: h1']
[34] []
[37] []
```



Conclusion

1. People are going to be much better at debugging than machines for the foreseeable future.
2. Efficiently capturing and sharing debug knowledge is the problem
3. All debug is a decision tree
4. New failures can be treated as permutations of previous failures enabling reuse of portions of decision tree collateral across disparate failures
5. People interact with data sources in very limited ways that can be abstracted to a tractable set of functions