2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

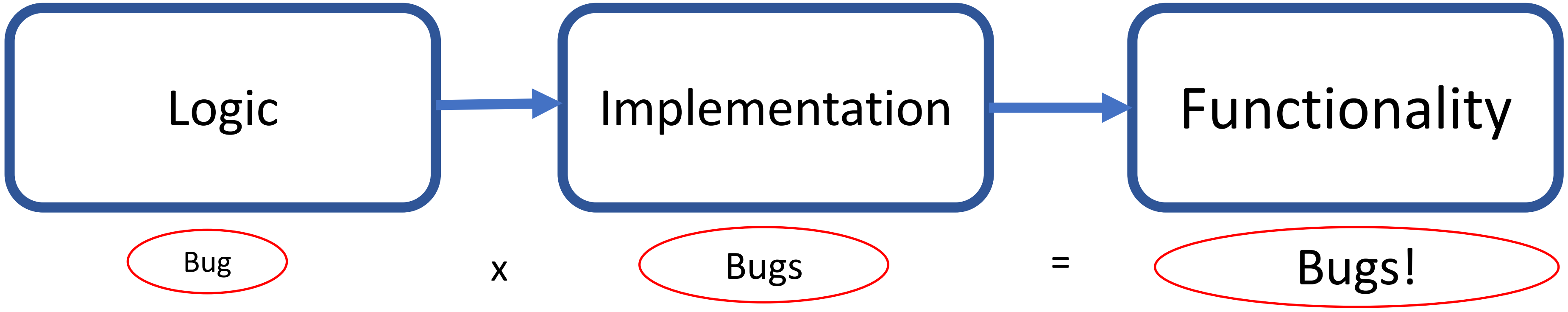
Tree Data Framework for Code Generation: Application of Generating UVM Testbench for Complex Designs

Chenhui Huang, Yu Sun, Divyang Agrawal

Tenstorrent Inc.

accellera SYSTEMS INITIATIVE

Why do DV engineers prefer generated code?



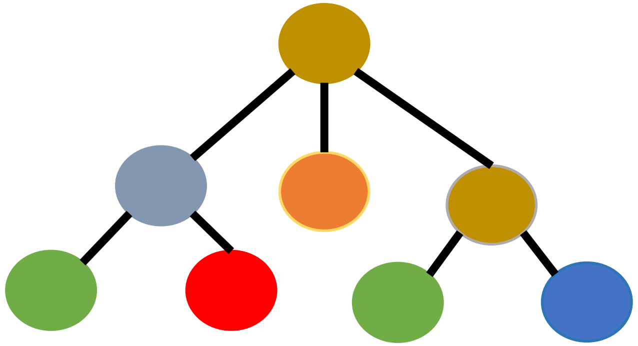
- Consistency and quality
- Fewer errors, faster turnaround on fixes
- Reduced time to production

Traditional code generation

- Macros as building blocks
- Templates where code can be inserted
- Configuration options to customize macros
- Code insertion – statically or dynamically

Inherent shortcomings of this approach!

Gingko framework

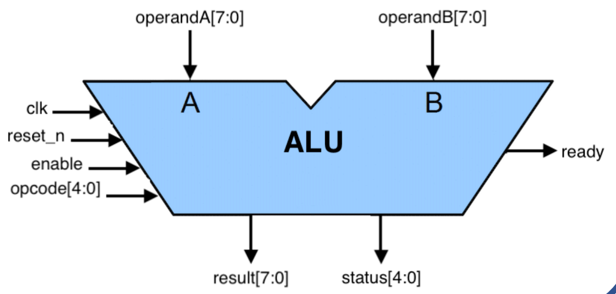


- Code generating framework
- Uses the tree data structure
- Applied to a UVM testbench generation flow, but language agnostic
- GenUVM is proof of concept application and can do more

SQL python django

How Gingko generates a UVM testbench

RTL Design



→

Gingko Framework

Collection Phase

Configuration Phase

Generation Phase

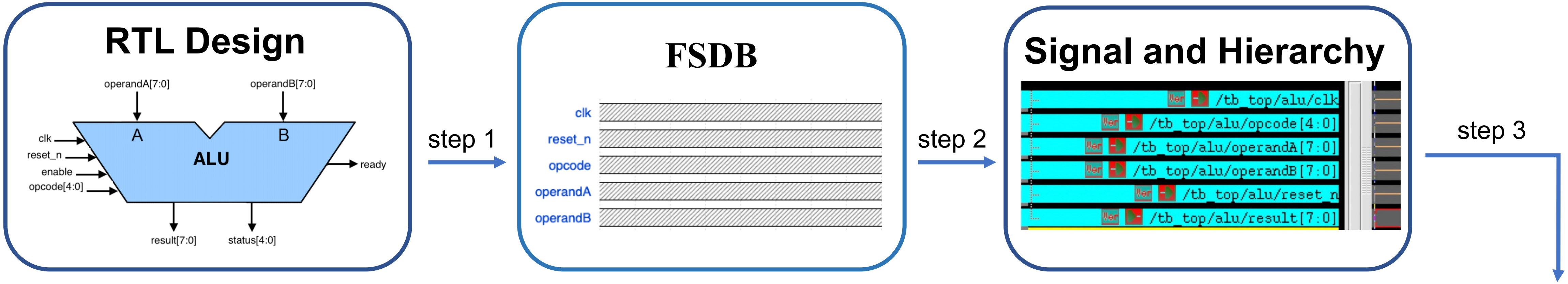
Database **Web GUI**

Python Lib

→

UVM Testbench

Collection Phase



Step 1: Generate the FSDB from design (single timestamp)
Step 2: Extract RTL signal and hierarchy information
Step 3: Upload and store in SQL database

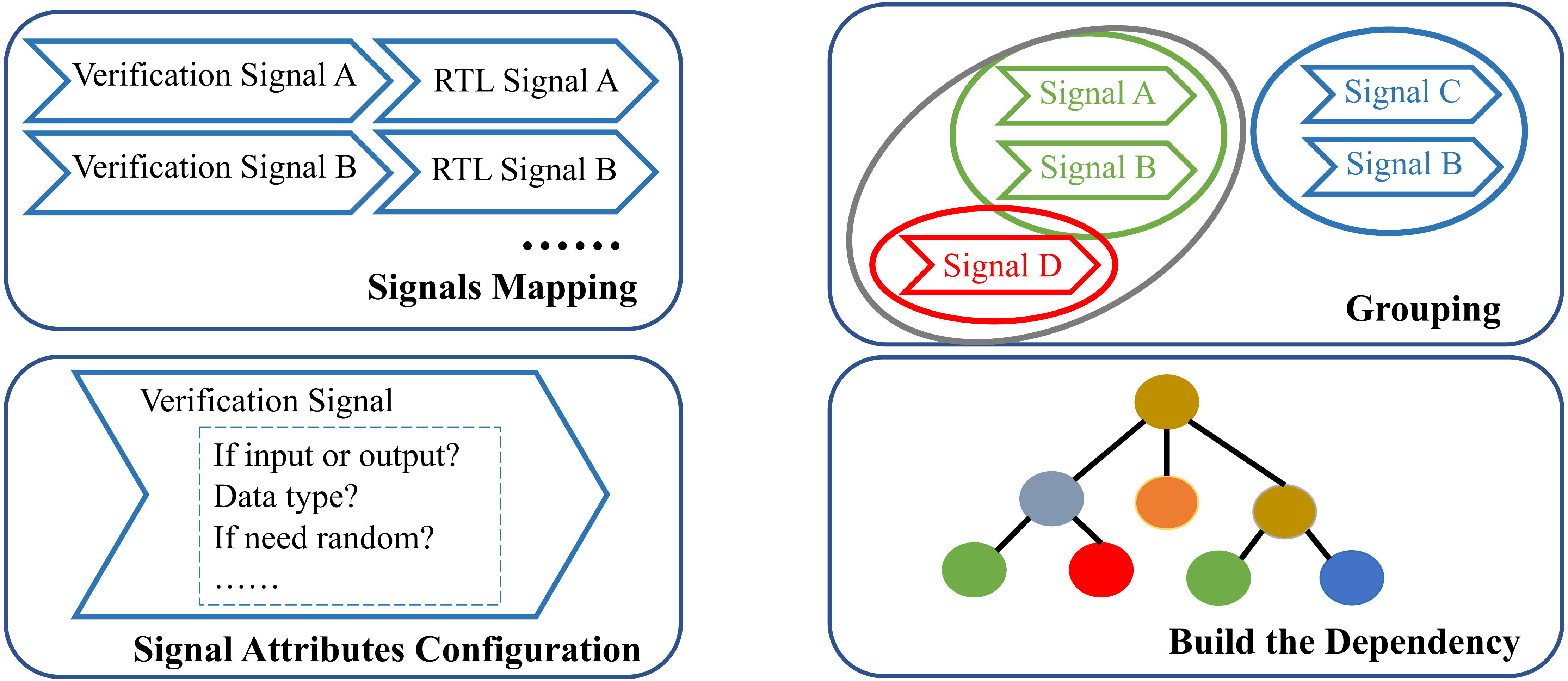
RTL database contents

Signal Property Extracted from FSDB

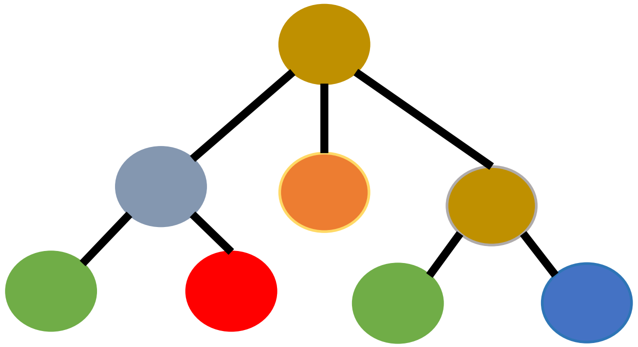
SHA Tag for each signal

vcd_type	type	hier_level	hier	signal_hier	signal_range	signal_name	signal_level	bit_range	l	r	num_bits	_rtl_signal_sha256
reg	output	3	tb_top.alu		result[7:0]	result	1	7:0	7	0	8	8907ead2e400f09500
reg	output	3	tb_top.alu		status[4:0]	status	1	4:0	4	0	5	69583eb16563277c4
reg	output	3	tb_top.alu		ready	ready	1		0	0	1	f11523034d577d60ea
wire	input	3	tb_top.alu		clk	clk	1		0	0	1	df1ff84801802fc0300
wire	input	3	tb_top.alu		reset_n	reset_n	1		0	0	1	0b04c011e3cc89e548
wire	input	3	tb_top.alu		operandA[7:0]	operandA	1	7:0	7	0	8	0eb5653a593ccaee6
wire	input	3	tb_top.alu		operandB[7:0]	operandB	1	7:0	7	0	8	1b71504aad5d0d6279
wire	input	3	tb_top.alu		opcode[4:0]	opcode	1	4:0	4	0	5	1fcc909a0321f4511b7
wire	input	3	tb_top.alu		enable	enable	1		0	0	1	8b1dd81265c6ff97077

Configuration Phase

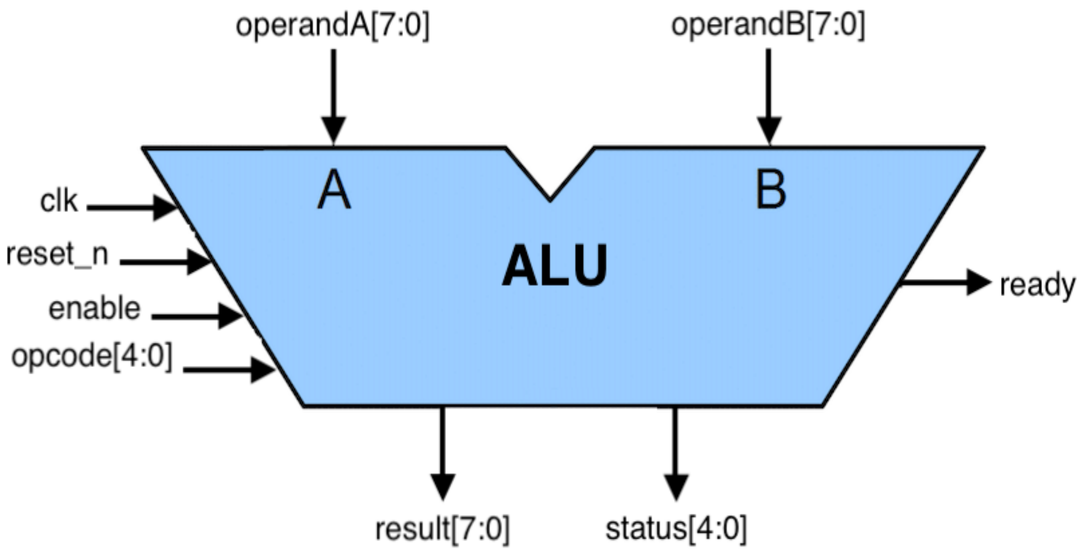


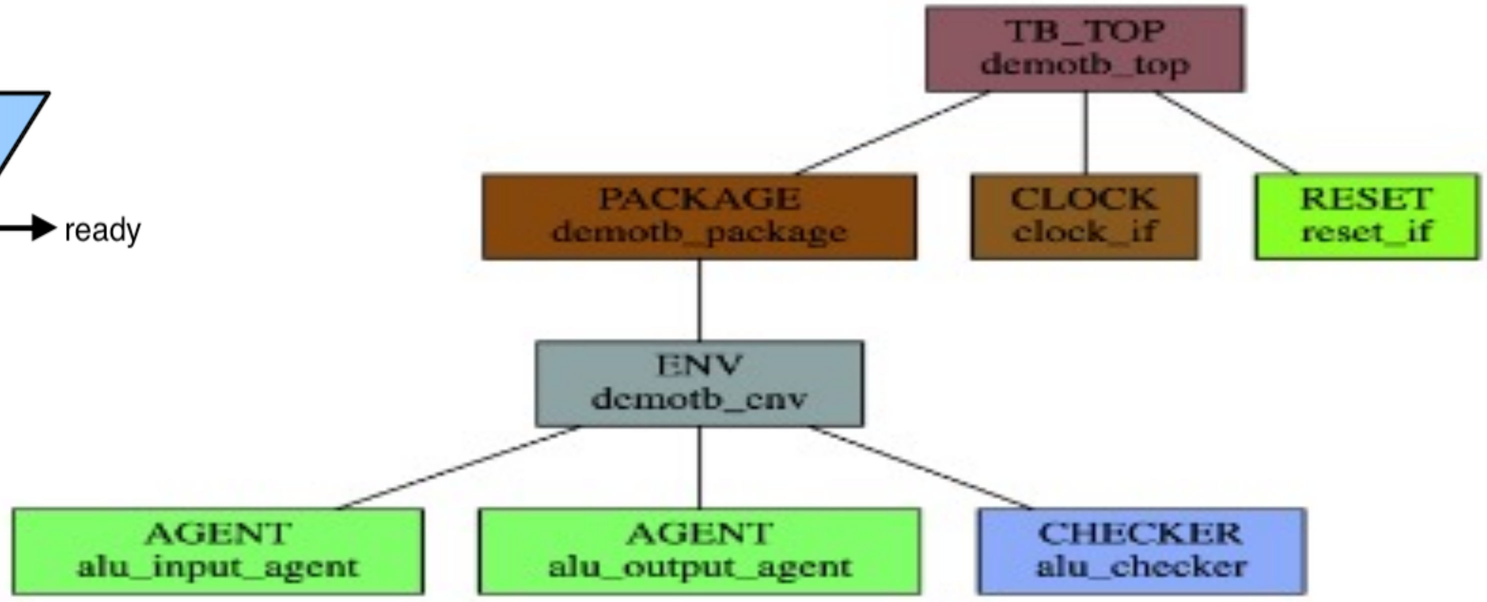
Component Element (CE) Type



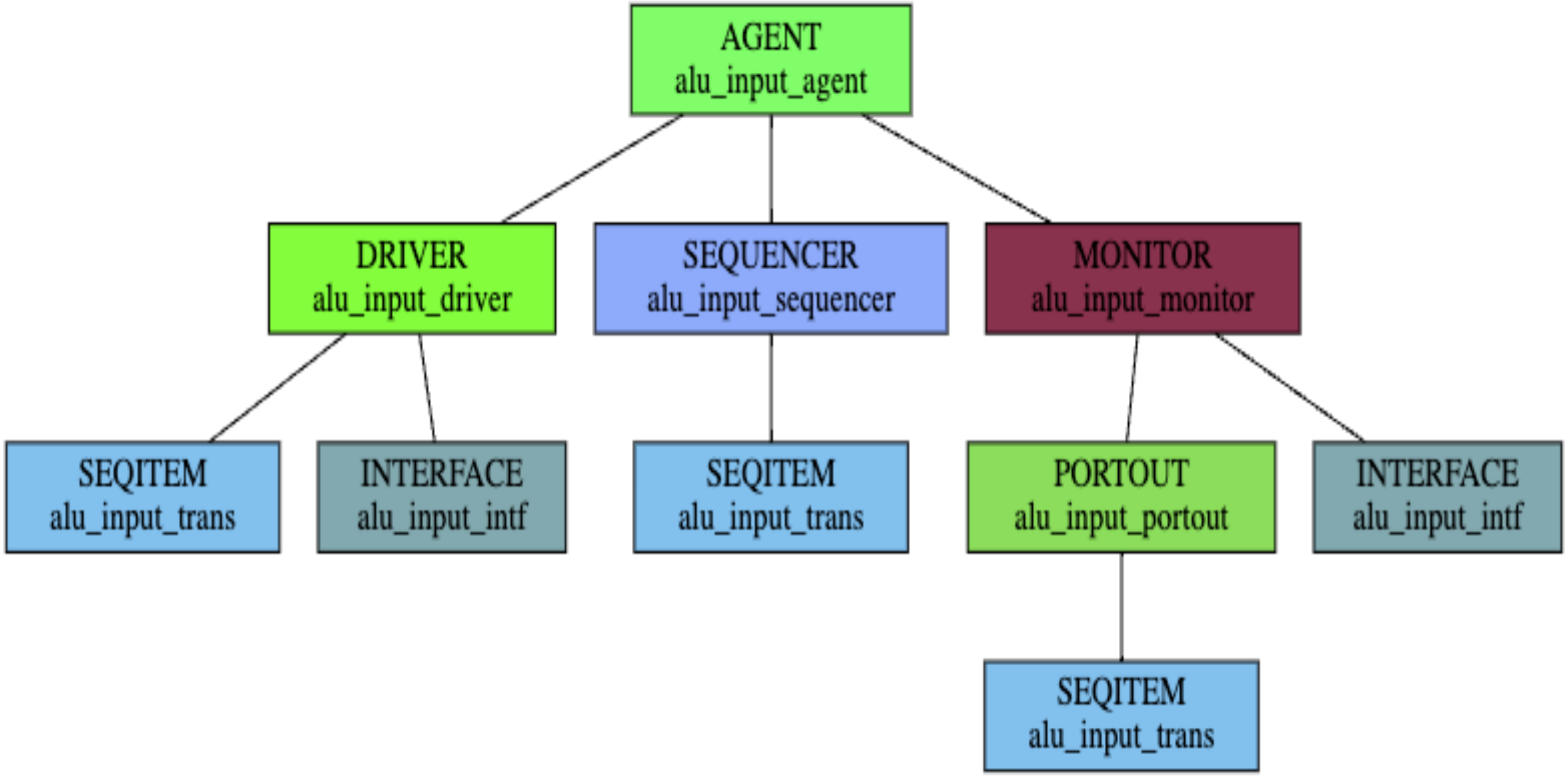
- Each tree node has one CE type
- Each CE type represents different functionality in the tree
- Each CE type has its Python callback handler code
- Handler code is called during the tree traversal in the generating phase
- Each CE type can have multiple pieces of handler code

Testbench structure

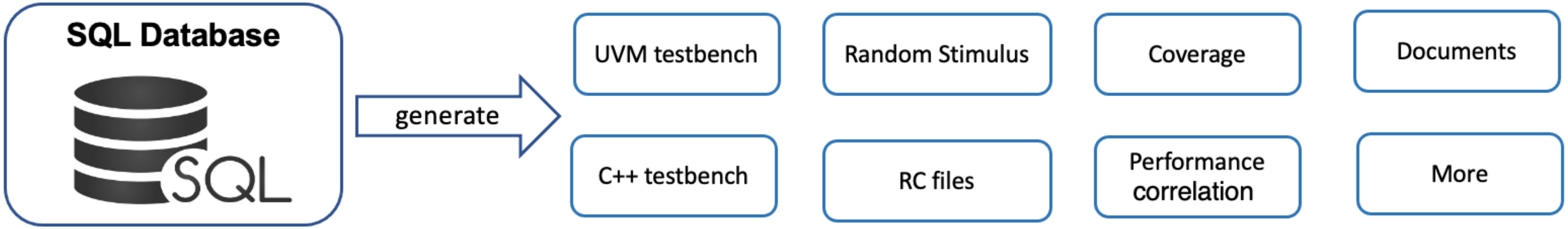




Tree data structure: Agent



Generation phase



Based on the CE type handler code, the framework can generate different targets while logic structure stays the same.

Why database?

- Separate data, logic and view instead of the traditional monolithic script to the code generation framework
- Retains history on design and testbench, enables reuse
- Foundation for enable AI based testbench generation
- Standard SQL API reduces the development cycle

Gingko: Advantages

- Testbench is independent of language, methodology and tools
- Flexible representation via the tree data structure
- Significantly less maintenance compared to metadata and template libraries
- Extremely fast turnaround on TB generation with changes in design
- Additional applications such as coverage, design quality analysis and more

Gingko: Constraints

- Less efficient for directed sequences
- Learning curve for DV engineers

Conclusion

- Tenstorrent has successfully deployed Gingko to generate UVM testbenches for various sub-units of a high-performance RISC-V CPU
- Significant learnings as engineers use Gingko leading to future work

Future work

- Extend usage of Gingko to generate more than UVM testbench
- Enhance the sequence model and diversify the stimulus
- Apply machine learning algorithm to generate testbench structure

Thank you for Listening

Questions