Shift left of IP Validation on Zebu

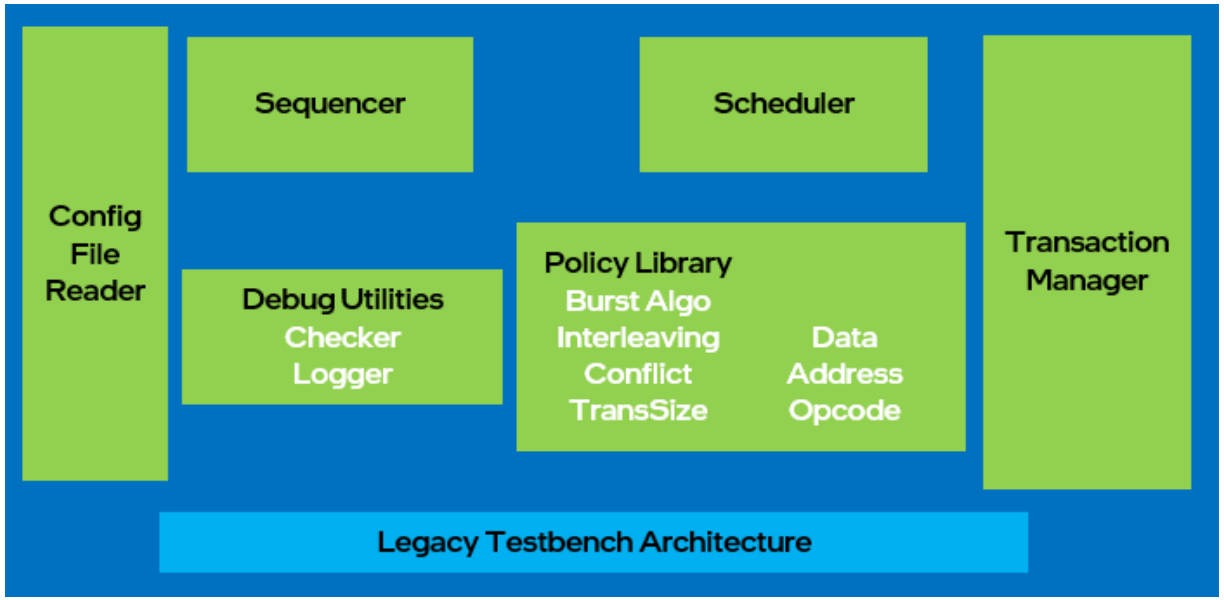
Intel Technology

Jagvinder Yadav
Aditya Chitrode

Agenda

- ❑Traditional approach towards Coherency validation in Emulation
- ❑New way forward : Industry standard UVM-SC methodology
- ❑Solving Multithreading challenges with UVM-SC
- ❑Randomization with CSOLVER
- ❑Debug Toolkit
- ❑Key Challenges
- ❑Takeaways

Coherency validation in Emulation : Traditional Testbench Architecture



TB Components (C++ based S/W TB)

Limitations of Traditional Methodology

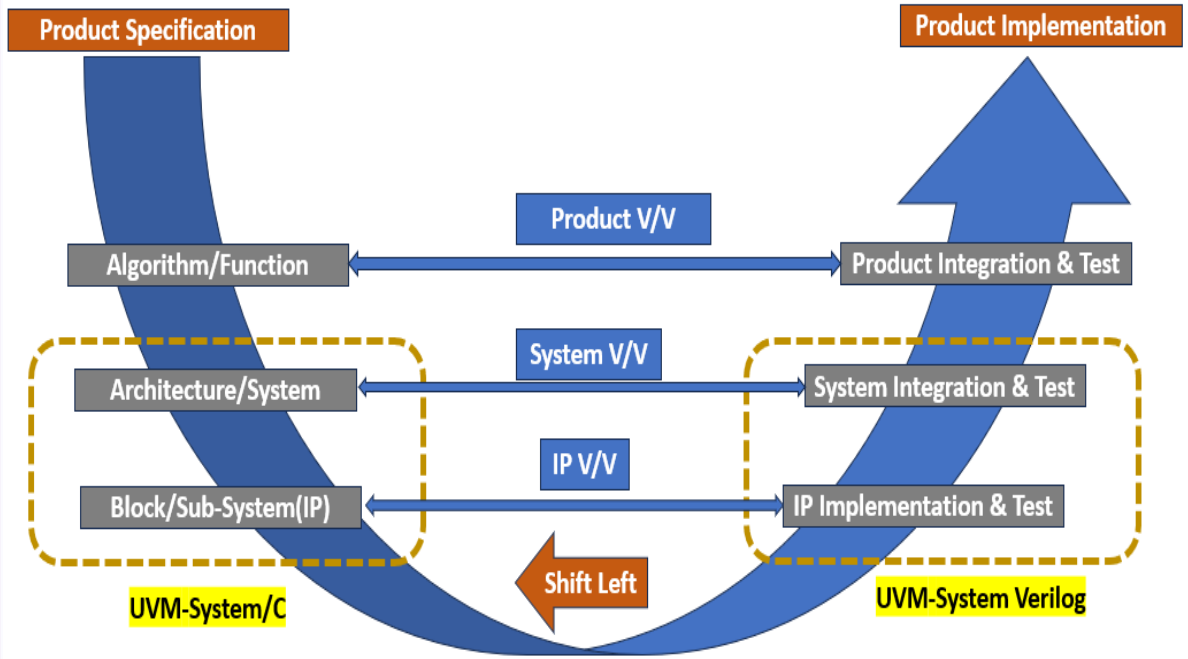
- ❑Intel proprietary methodology
- ❑Lack of constraint-solving technology
- ❑Limited scheduling controls
- ❑Lack of cross-scenario interactions
- ❑Scenario reproducibility challenges

The limitations & challenges with traditional methodology prompted us to migrate to UVM-SC methodology

New way forward : Industry standard UVM-SC methodology

Benefits offered by UVM-SC over Traditional approach

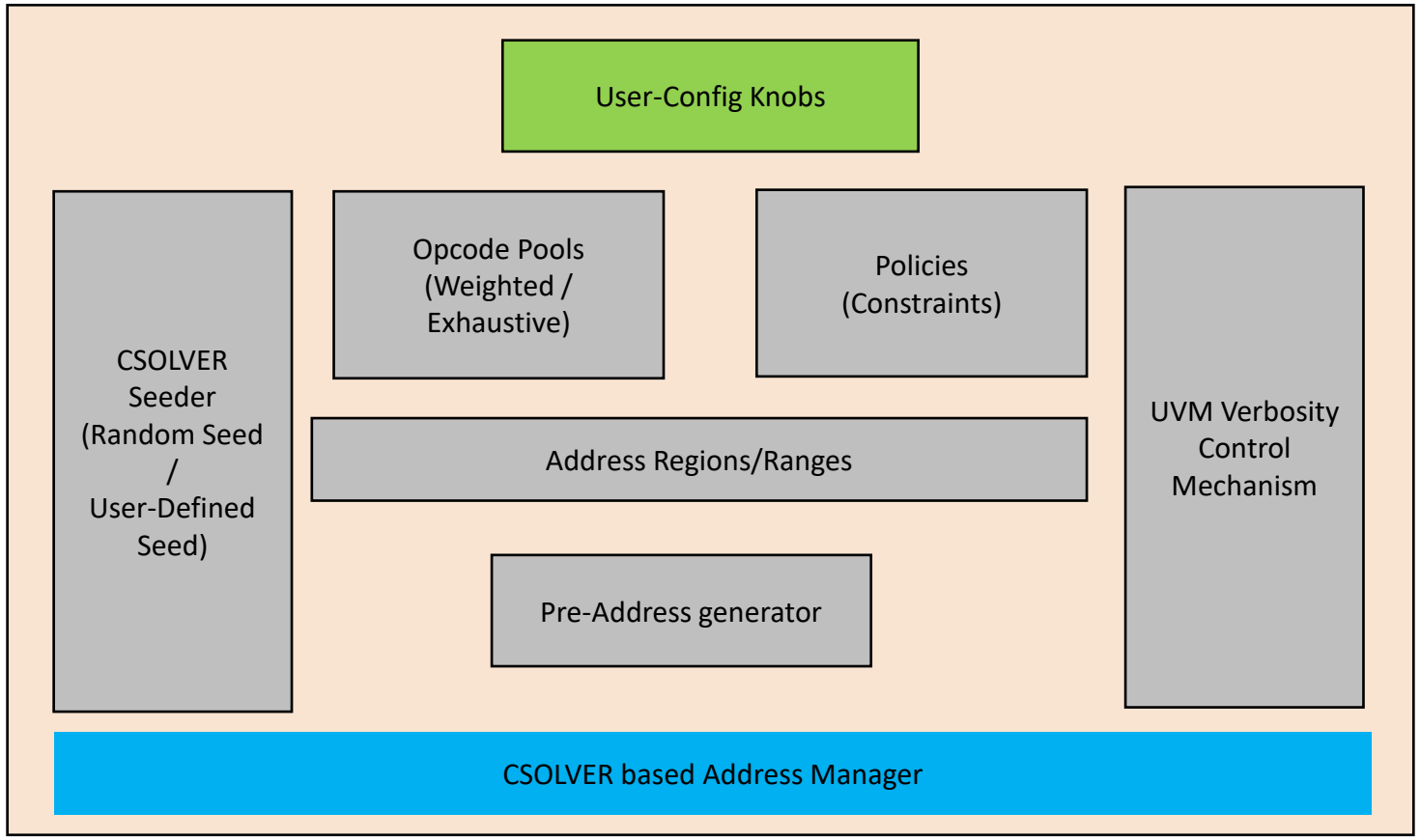
- ❑Accellera's open-source & a proven industry standard
- ❑Highly configurable, modular & scalable methodology
- ❑Reusability : block level to system-level verification
- ❑Reduce/eliminate methodology barriers b/w simulation & emulation



Shift Left : Early Validation & Verification in a standardized way

UVM-SC capabilities enable testbench & content bring-up in lock-step with simulation and truly achieve left-shift way.

Solving randomization challenges with CSOLVER



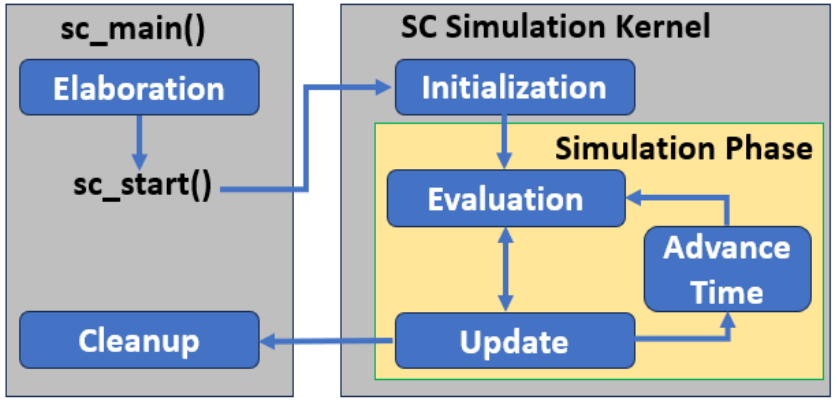
Powerful CSOLVER features exercised for UVM-SC Emulation Testbench

- ❑ Pre/Post randomization()
- ❑ Randomization Controls
- ❑ Random seeding - Global/Random class instance level
- ❑ Constraints Controls
  - Soft Constraint
  - Variable ordering Constraint (Solve-before)
  - Distribution Constraint
  - Uniqueness Constraint
  - Implication Constraint
  - Foreach Iterative/Nested Constraint
- ❑ Solver Tracing

CSOLVER is interoperable with SV-SystemC, gives better performance and helps in developing scalable, reusable constrained-random testbenches for validation needs in emulation

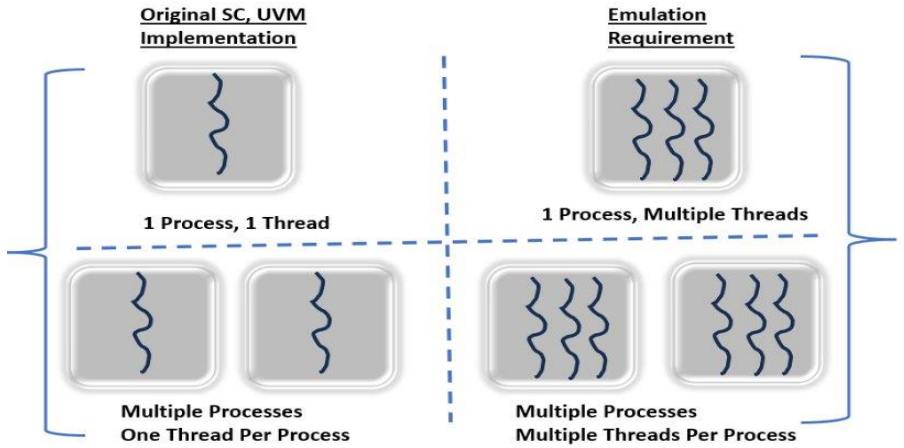
Solving Multithreading challenges with UVM-SC

Single-Threaded UVM-SC Architecture



- Not thread-safe (due to usage of static data)
- Cooperative multithreading for scheduling & computation

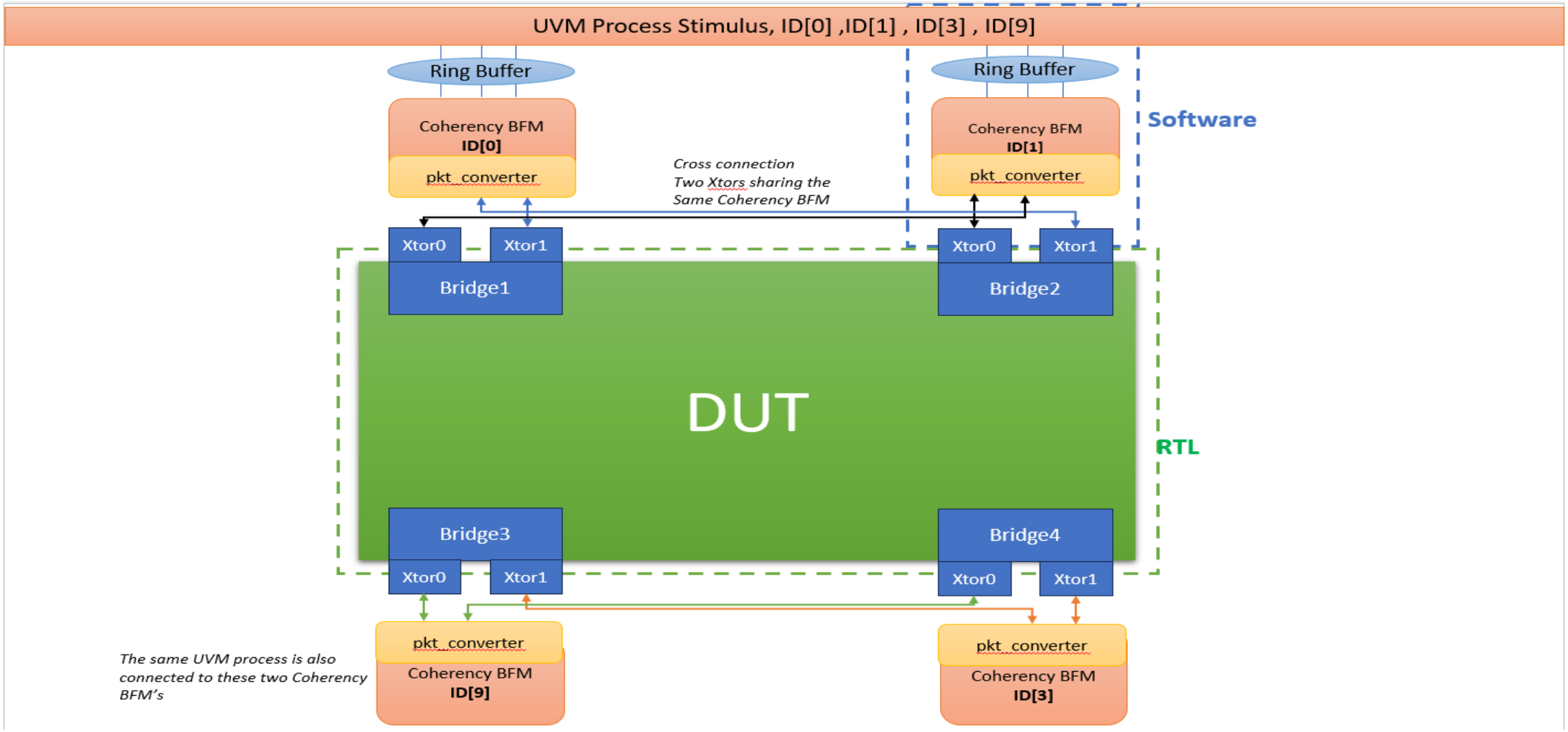
Multi-Threaded UVM-SC Architecture



- Coarse-grain multi-threading support for SC
- Shared-buffer based multi-process communication

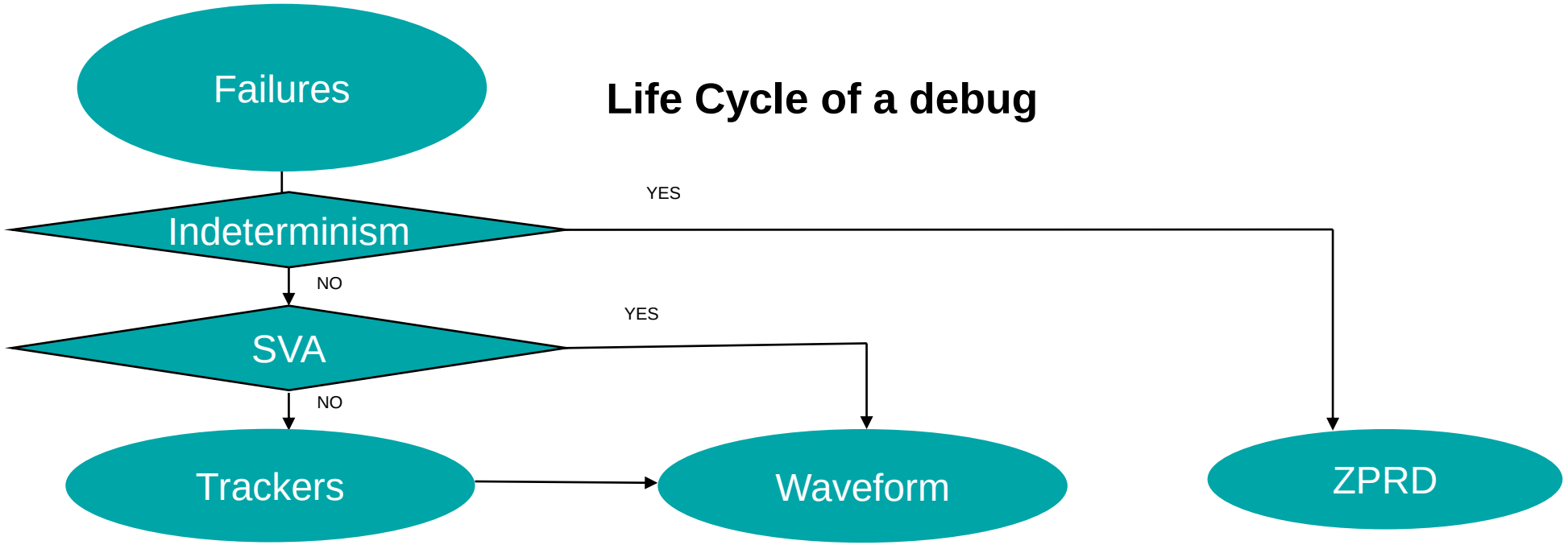
Multi-threading solution helped in executing stimulus computations on very fast cores in a separate process while the host continues run more of its threads in parallel

DUT Testbench Setup



Debug Toolkit

Features	UTF Switch	Use case
Trackers	NA	Internal trackers for different interfaces to trace the life cycle of the transaction.
SVA	utf assertion_synthesis -enable ALL -verbose true	Synthesizable assertions and trigger-based W/F dump to narrow down the point of failure quickly.
ZPRD	debug -offline_debug_params {INCL_XTORS=true} ztopbuild -advanced_command {zoffline_debug -enable yes}	ZPRD enabled for the failure window and in some cases for the entire window to debug the failure.



Key Challenges

- **Performance V/s Reproducibility**
  - XTOR coded with performance intent (streaming DPIs)
  - Streaming causing reproducibility issues due to a multiple-xtor instances
  - Seeding-reseeding mechanism failing due to change in DPI ordering
- **Proposed Solution**
  - ZEBU support for global DPI ordering

Key Takeaways

- Adoption of industry standard UVM SC Methodology
- Multi threading with UVM SC
- Debug toolkit - Trackers, SVAs and zprd
- Performance v/s reproducibility


THANK YOU
Our Technology, Your Innovation™