



# Challenges in using UVM at SoC level

---

**Rohit Jindal**

**ST Microelectronics**

- General Overview of UVM
- Challenges in using UVM at SoC level
- Proposal for using UVM at SoC level

# What is UVM?



## *Universal Verification Methodology*

- Testbenches for HDL designs
- UVM has SystemVerilog Base Class Library
- Based on constrained Random Verification approach
- Industry standard developed by Accellera
- Apache 2.0 open-source license

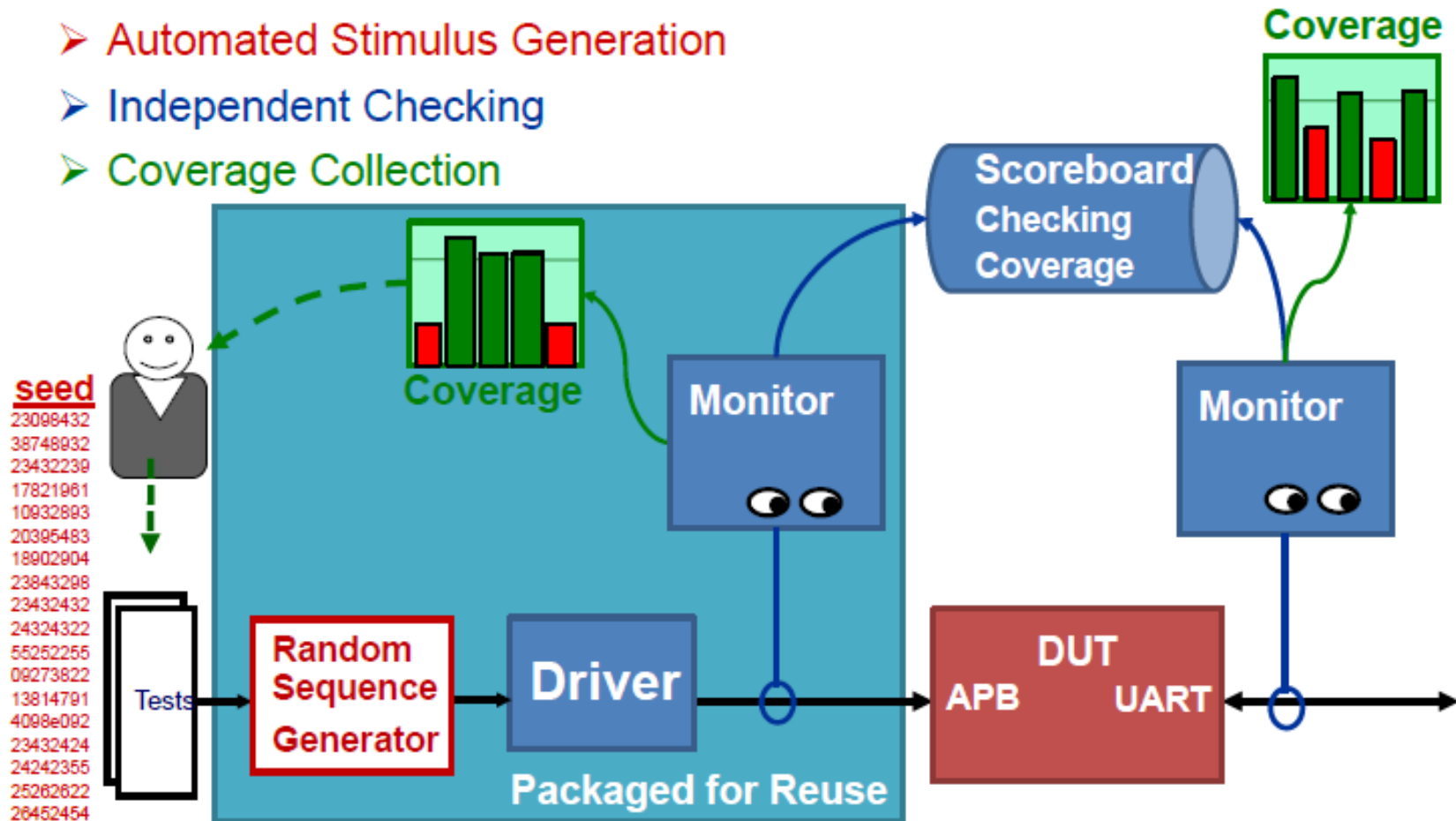
- Key features of the methodology:
  - Data design and stimulus generation
  - Building and running a verification environment
  - 3C : Checker, Coverage and Constrains

# The Goal : Automation



- Coverage Driven Verification (CDV) environments

- Automated Stimulus Generation
- Independent Checking
- Coverage Collection

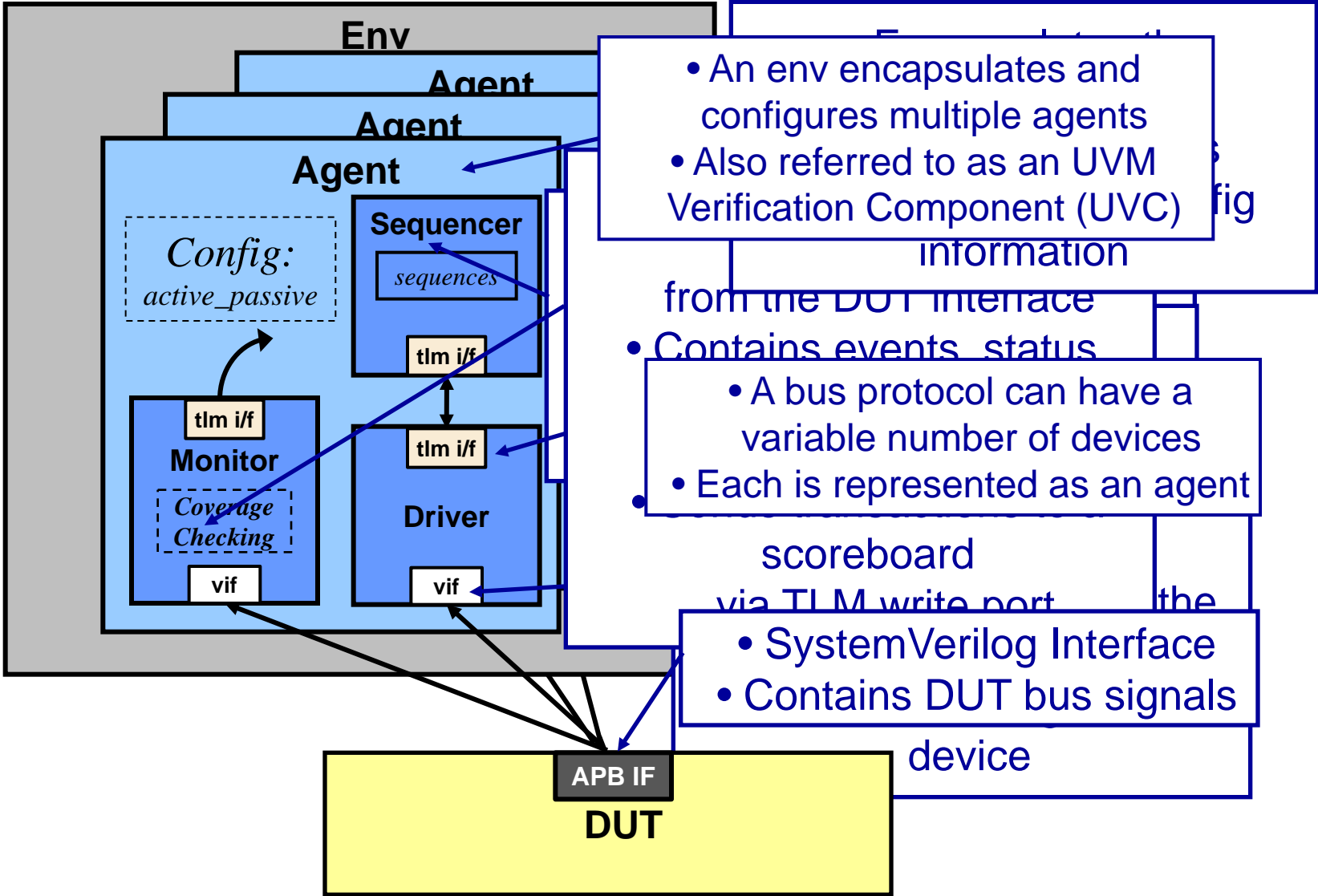


Source: Accellera DAC Presentation

- **Built-in automation**
  - Transaction manipulation (Print / Pack / [Deep] Copy / Record)
- **Defined testbench build and hook-up mechanism**
  - Flexible automatic test phase interface
  - Lot of phases defined for synchronization
- **Powerful stimulus generation mechanism using sequences**
  - On-the-fly randomization control
  - Sequence Layering & nested sequence
  - Virtual sequences [Controlling multiple sequencers from 'central place']
- **TLM communication**
  - Modular and Language independent
- **Supports reuse**
  - Module-to-system
  - Project-to-project

- **Sequences or Testcases**
  - UVM Tests or sequences separated from TestBench environment
- **Uniformity**
  - Standard structure for all Testbench components
- **UVM factories**
  - UVM factories give flexibility
  - Easy to update or modify components without changing original code
- **Configuration**
  - UVM configuration helps in customizing the environment from Top or anywhere
- **And more...**
  - Messaging utilities with on-the-fly control over verbosity
  - Ensures random stability
  - Compile once and run many times using snapshot

# Reusable UVM Component Architecture

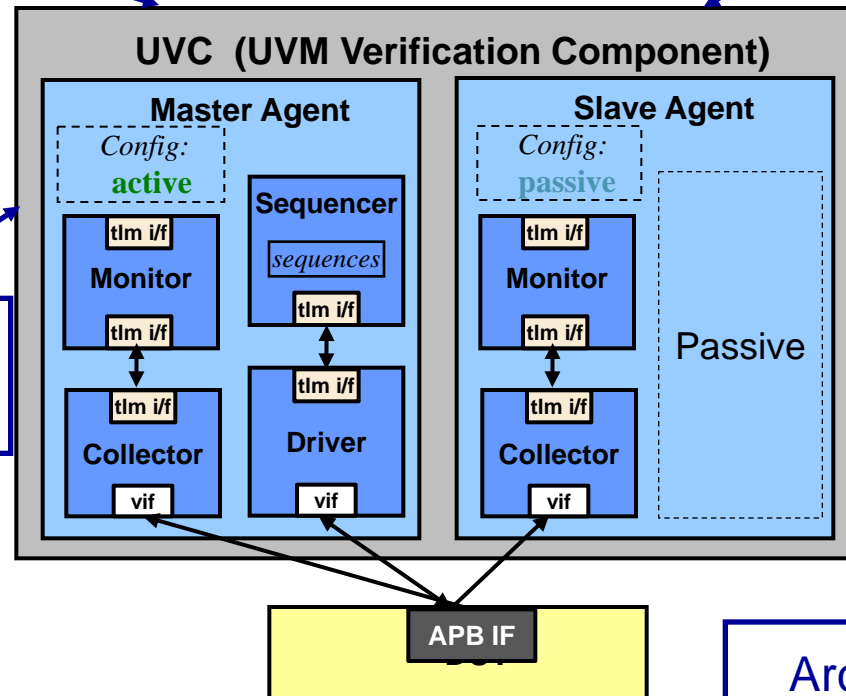


# What Are the Benefits Of This Architecture?



Encapsulating components makes it modular and simplifies maintenance

“Virtual” sequences enable control at the system level



Control and configure from above

Supports module-to-system reuse

Easy-to-use test writer interface

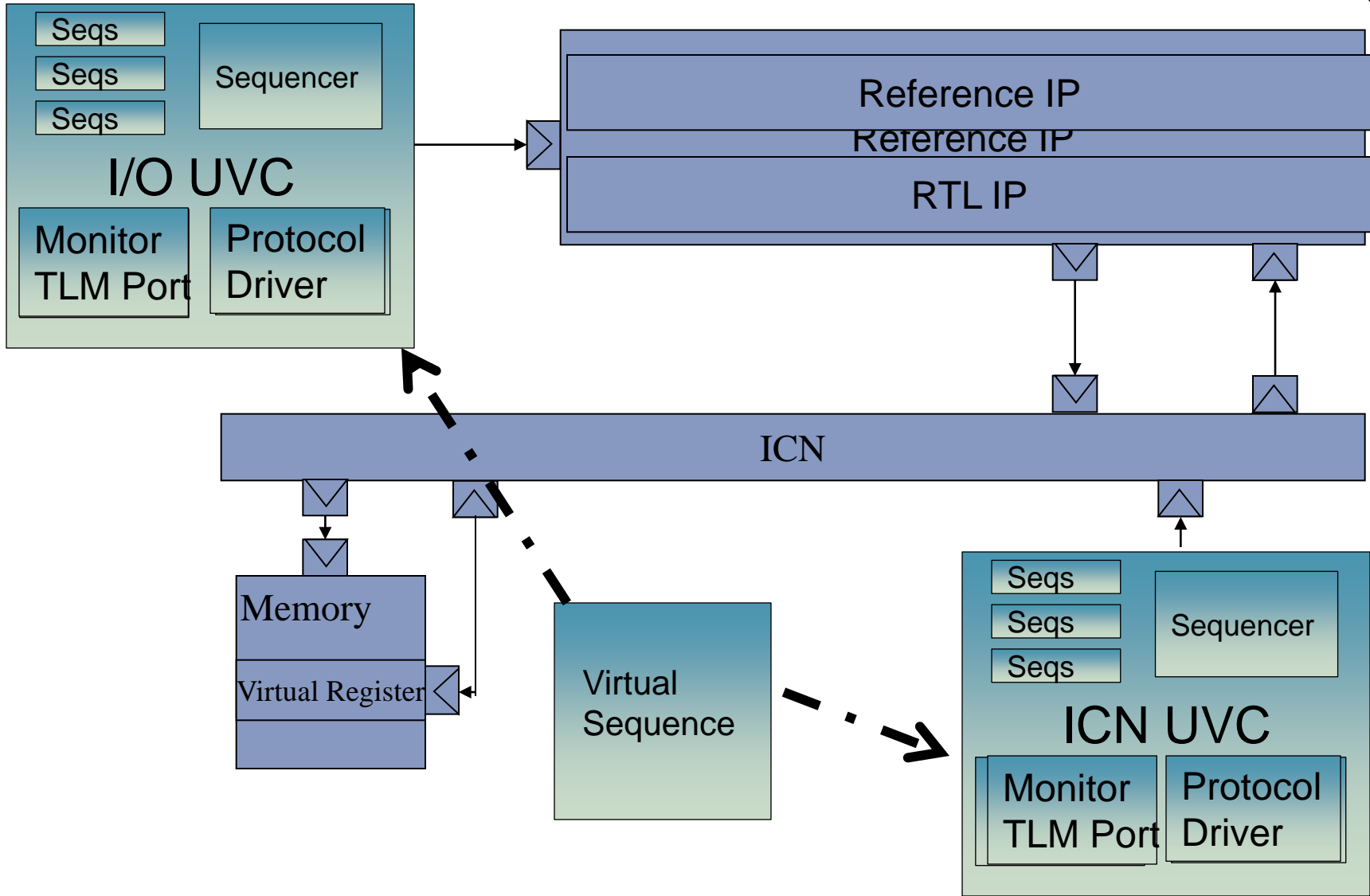
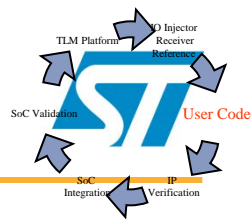
Architecture exactly the same across all languages!

**Each component has an UVM parent class with built-in utilities and automation ...**

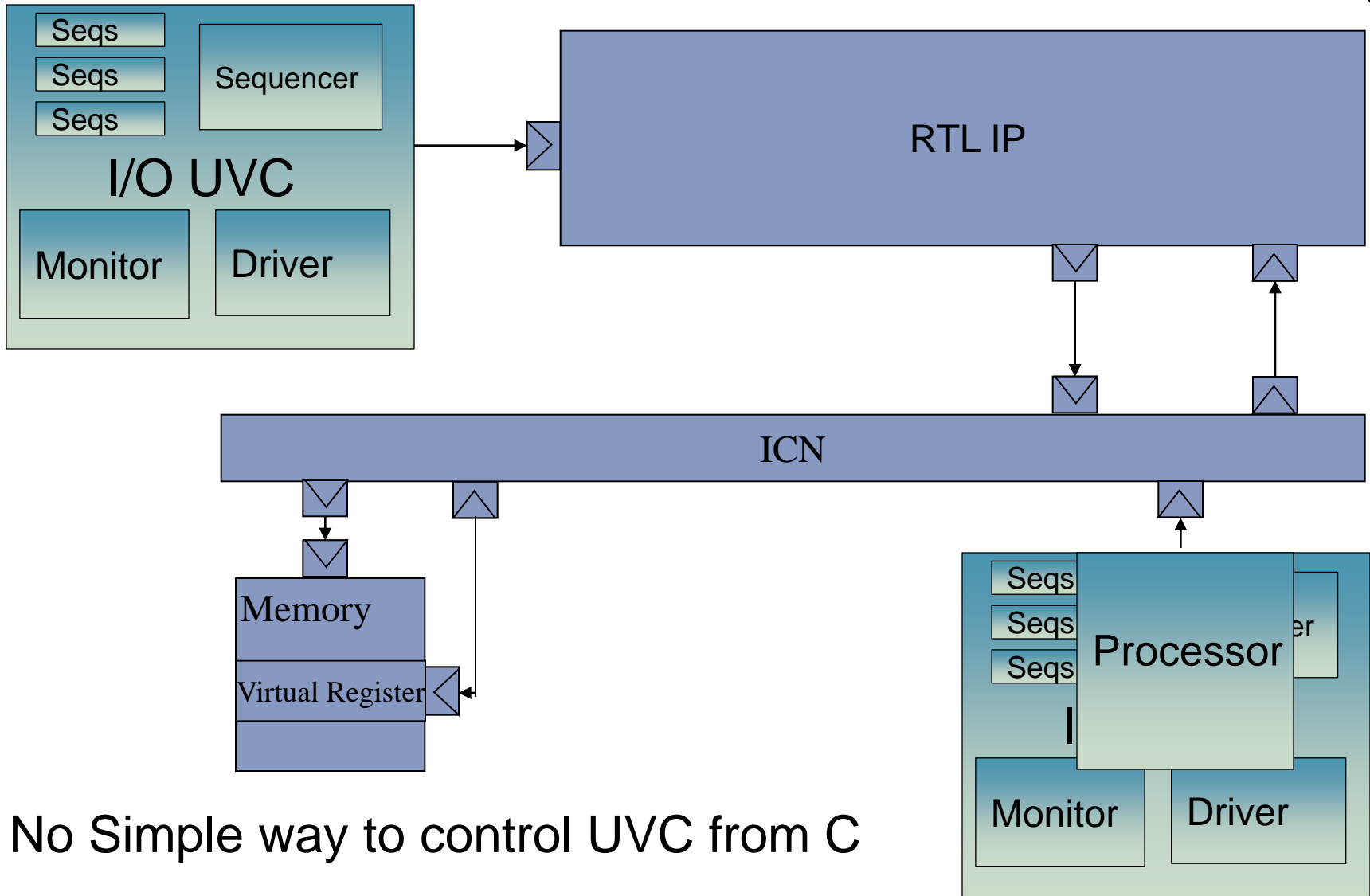
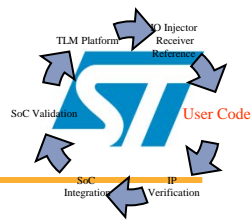
- General Overview of UVM
- Challenges in using UVM at SoC level
- Proposal for using UVM at SoC level

- ✓ Quick setup of SoC verification env (integration of all verification component)
- ✓ Synchronization or Control of verification components
- ✓ More emphasize is on system scenarios/ integration rather than on randomization
- ✓ Should able to reuse configuration of IP from IP or sub-system testcases
- ✓ Should able to re-produce validation issues

# UVM Verification Flow



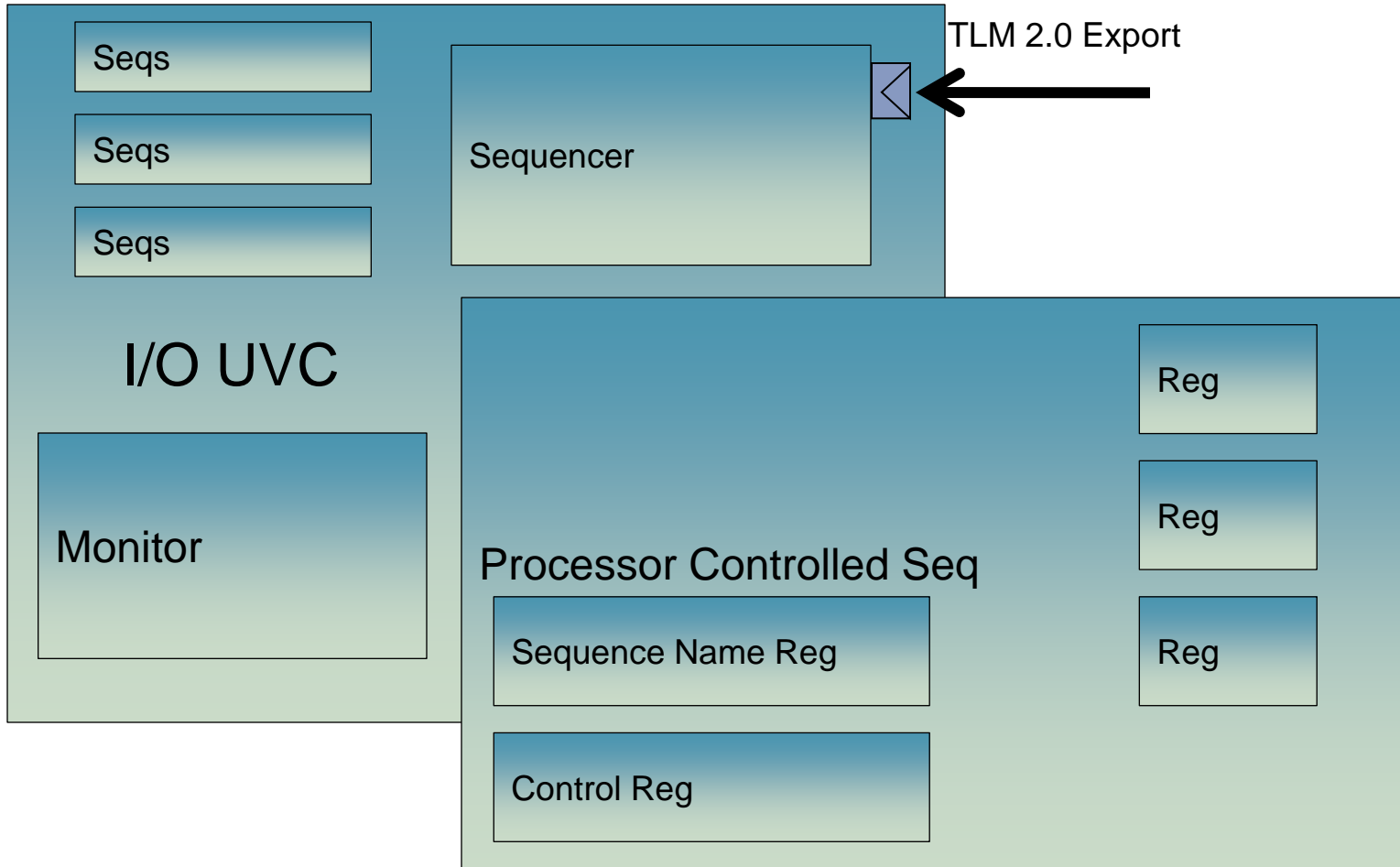
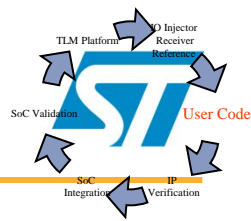
# UVM Verification Flow at SoC level



No Simple way to control UVC from C

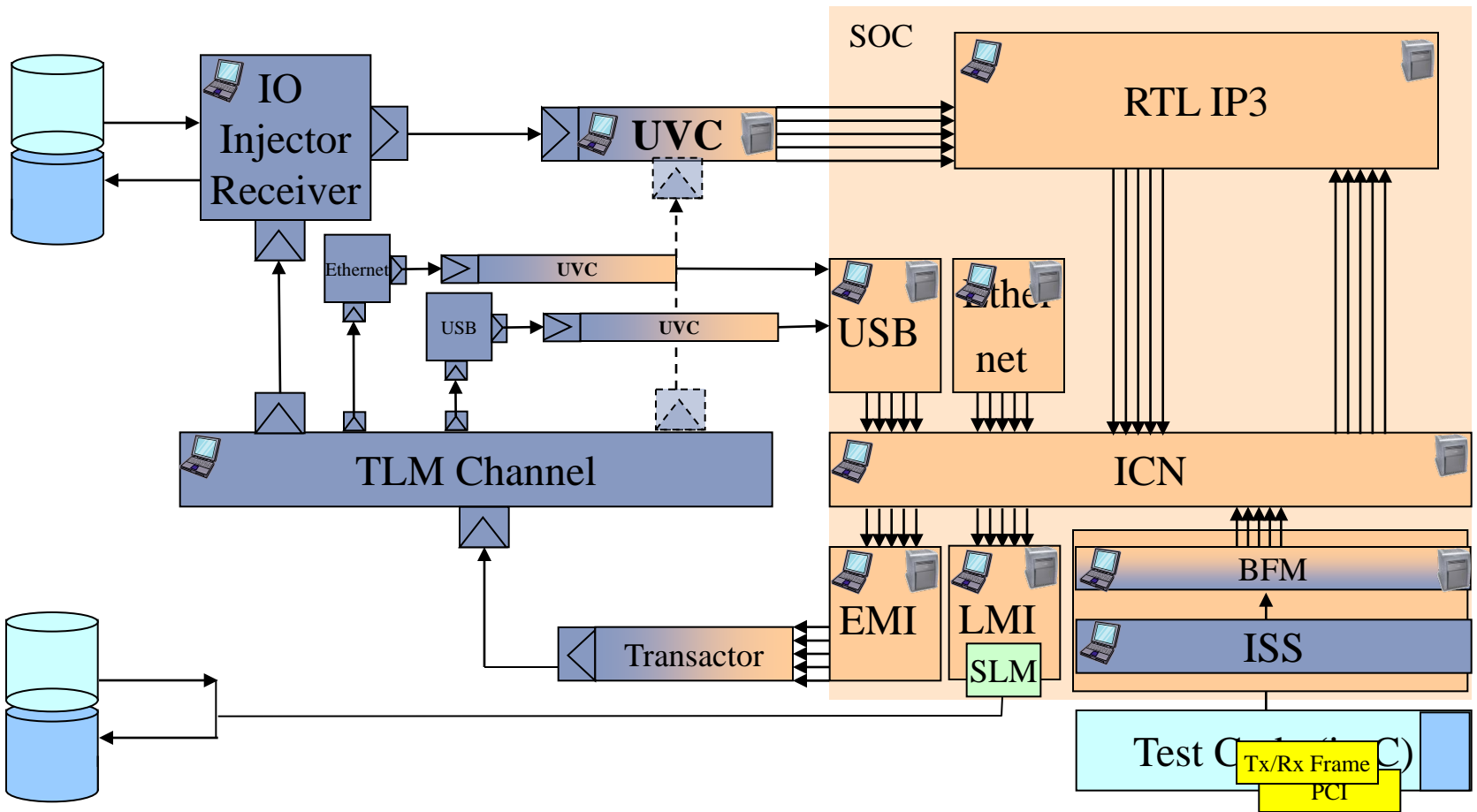
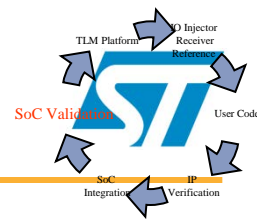
- No good way to control UVC from C or processor
- No standard way to support multiple language (available from Cadence but not Accellera or IEEE standard)
- Easier said than done to reuse the test cases (sequences) from IP->sub-system->SoC.

# Proposed UVC Architecture...



- Developed a wrapper to connect TLM port between SystemC and UVM
- Added TLM export in sequencer
- Added a sequence with registers which can be controlled through TLM port
- Developed Test platform to demonstrate controlling of sequence from TLM IP

# SoC Verification with UVM



TLM
  RTL
  User Code
  Transactor (Sc or SCEMI)

- ✓ Connection of EMI interface with SystemC TLM channel
- ✓ Connection of TLM SystemC port with UVM port
  - Developed Test Platform to demonstrate controlling of sequence from Processor
- ✓ C code running on ISS, which is writing to a TLM Port about the sequence information
- ✓ Based on the received information, UVC sequence runs the required sequence or generate the random/directed traffic
- ✓ Checked the generated data through C testcase

- UVM good at IP level and best with standard interfaces like AHB, AXI etc
- Best for constraint randomization verification
- Good in reusing verification components (like driver, monitor or agents)
- For efficient reuse verification engineers need to be expert in UVM and OOPs concept
- Still some gray areas for how to use UVM at SoC and Accellera can work in this area

- Seamless connection between UVM and SystemC ports
- TLM export in sequencer library
- Sequence with registers which can be written from TLM port in sequencer
- Event queue in sequencer for synchronization
- Standard name for some of the sequences like `generate_interrupt`, `clear_interrupt`, `dma_data_transfer` etc

---

# Thanks