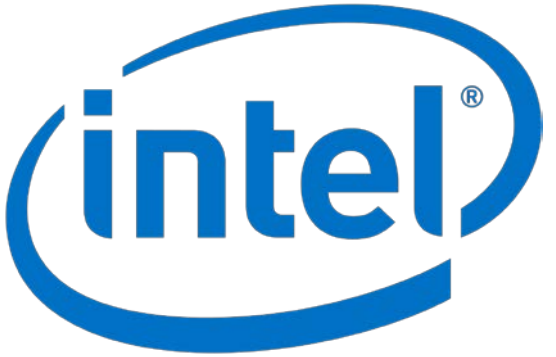
2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Automated Generation of RAL-based UVM Sequences

Satyajit Sinari
Timothy McLean
Benjamin Applequist
Vijayakrishnan Rousseau
Geddy Lallathin



accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

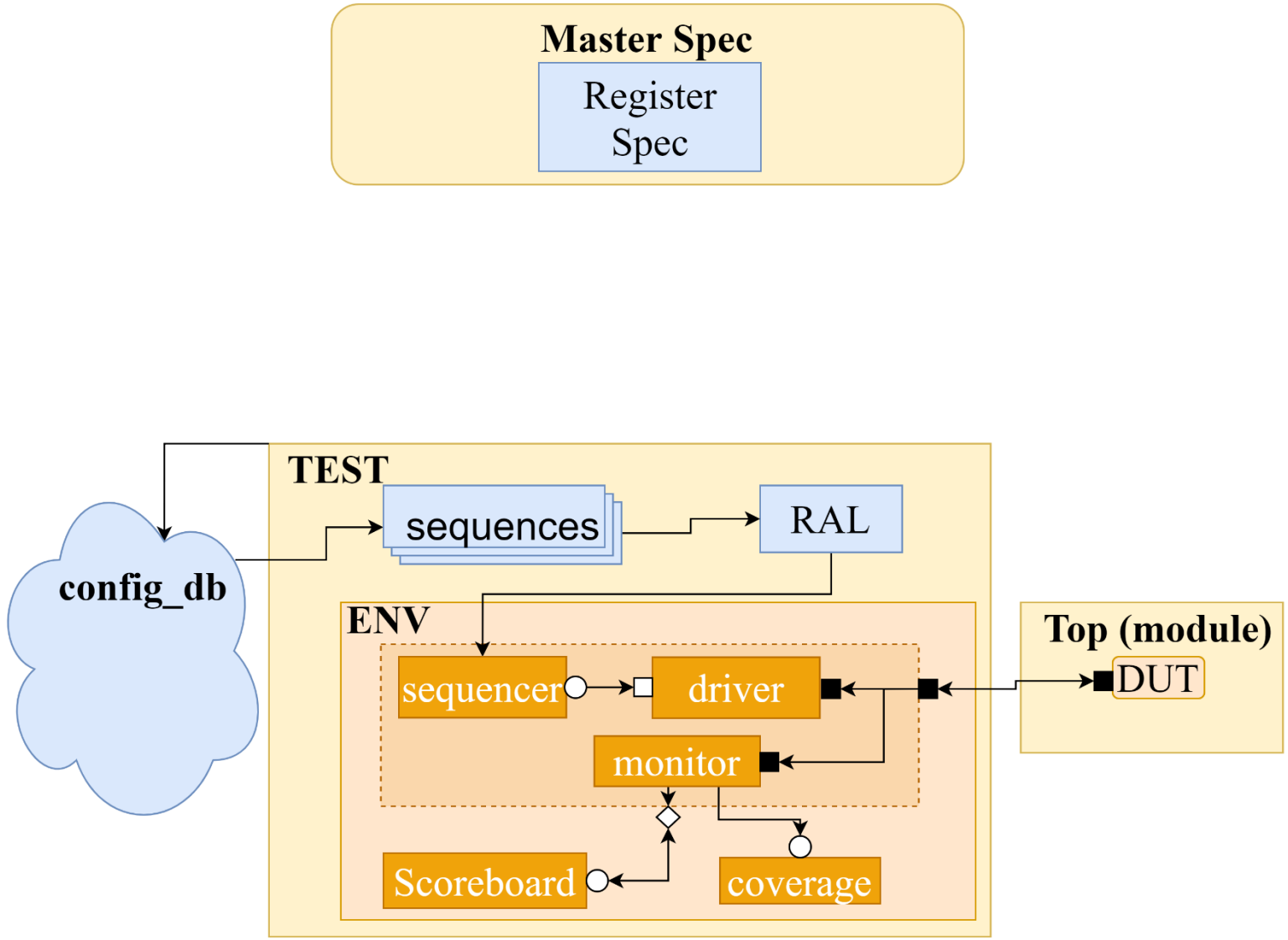
Overview

- Original Validation Model using UVM
- Problems with the Original model with complex designs
- Proposed Scalable Automated Validation Model
- Implementation with an example

accellera SYSTEMS INITIATIVE

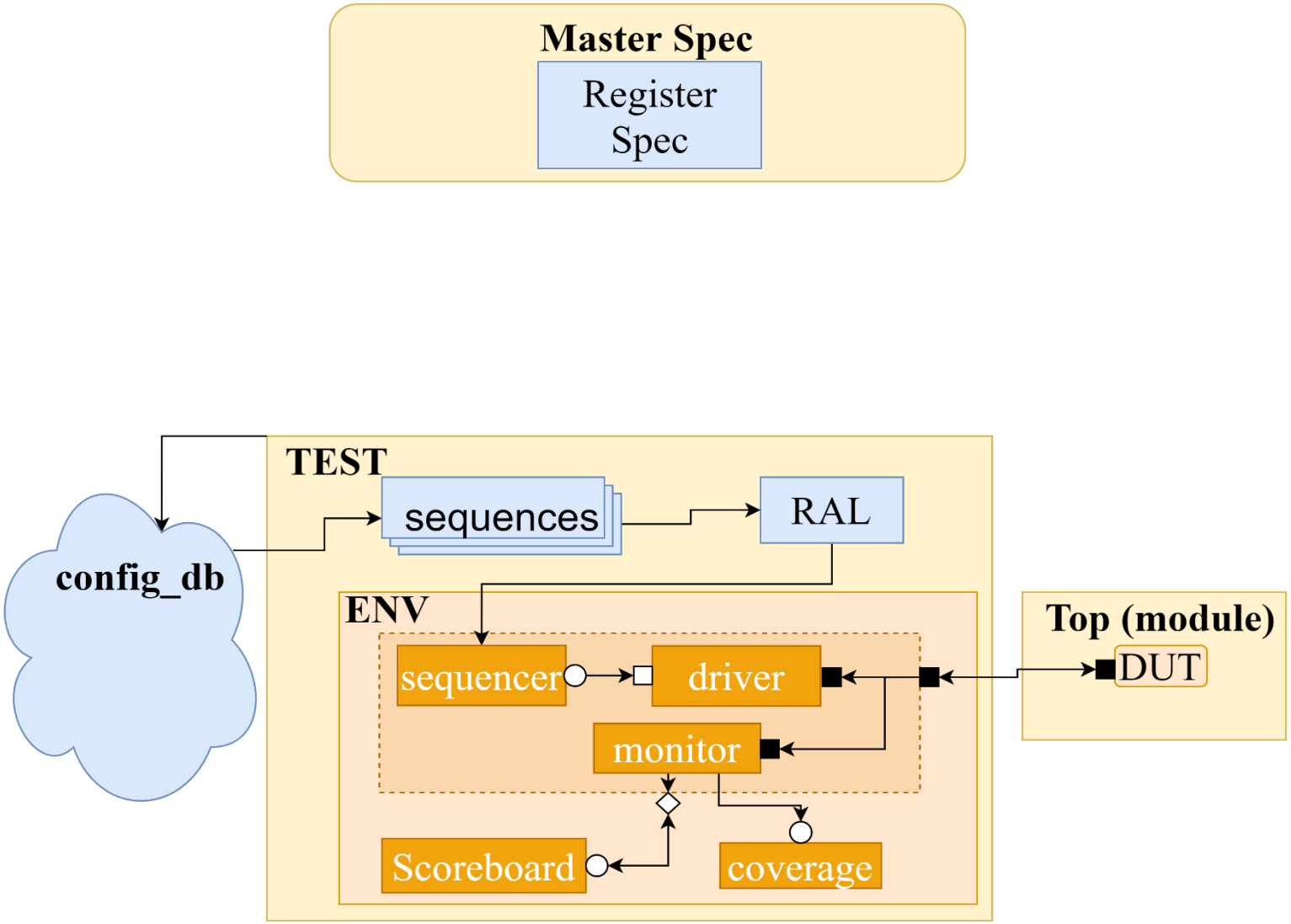
Original Validation Model

- Architectural master specifications
- Testbench using industry standard UVM architecture
  - Env, Agent, Driver, Monitor, etc.
  - UVM sequences(manual)
  - RAL (Register Automation Layer)



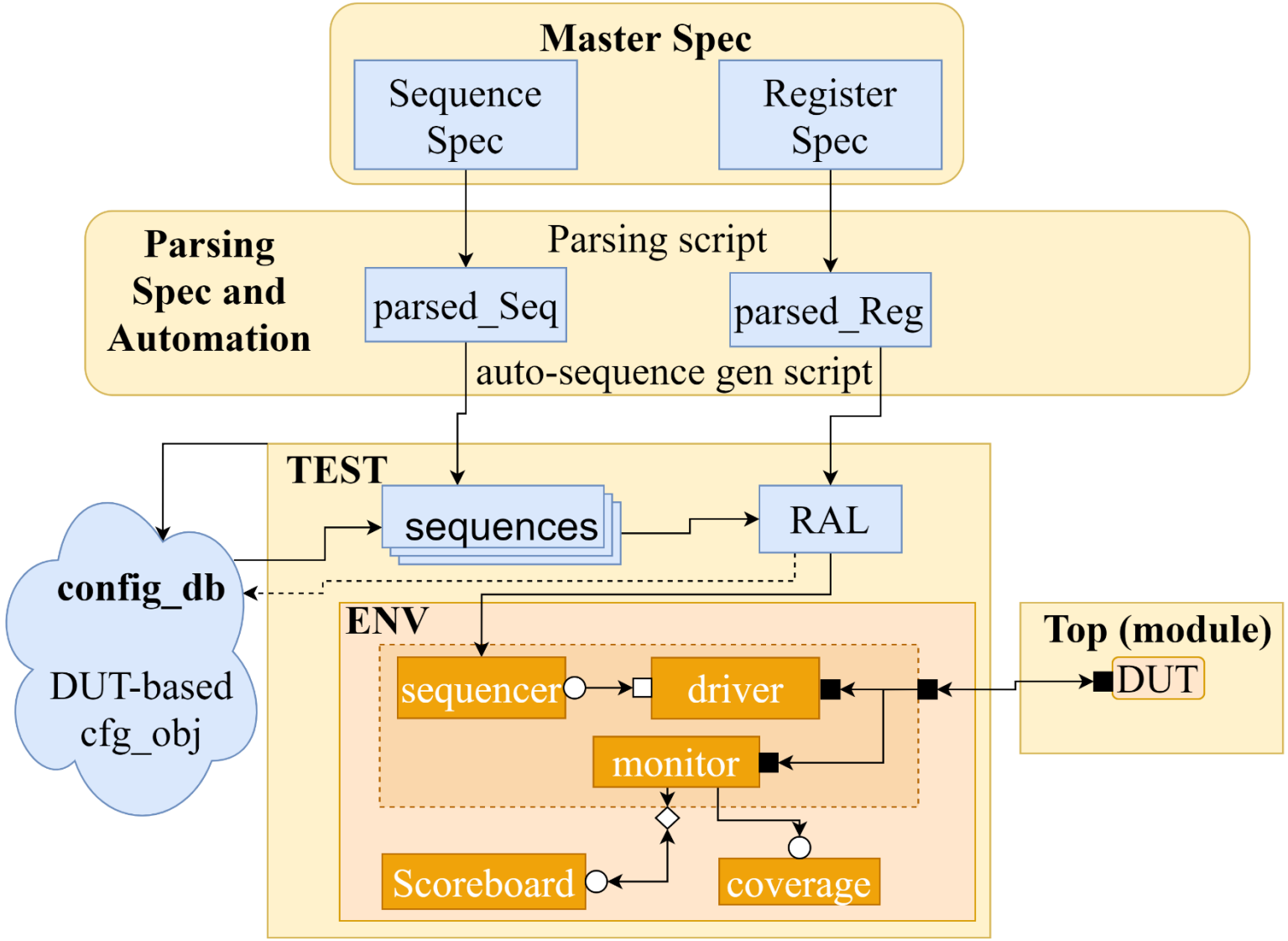
Problems with the orig model

- Large and complex Hardware Designs
  1. Maintaining manually generated components becomes hard
  2. Issues during randomization



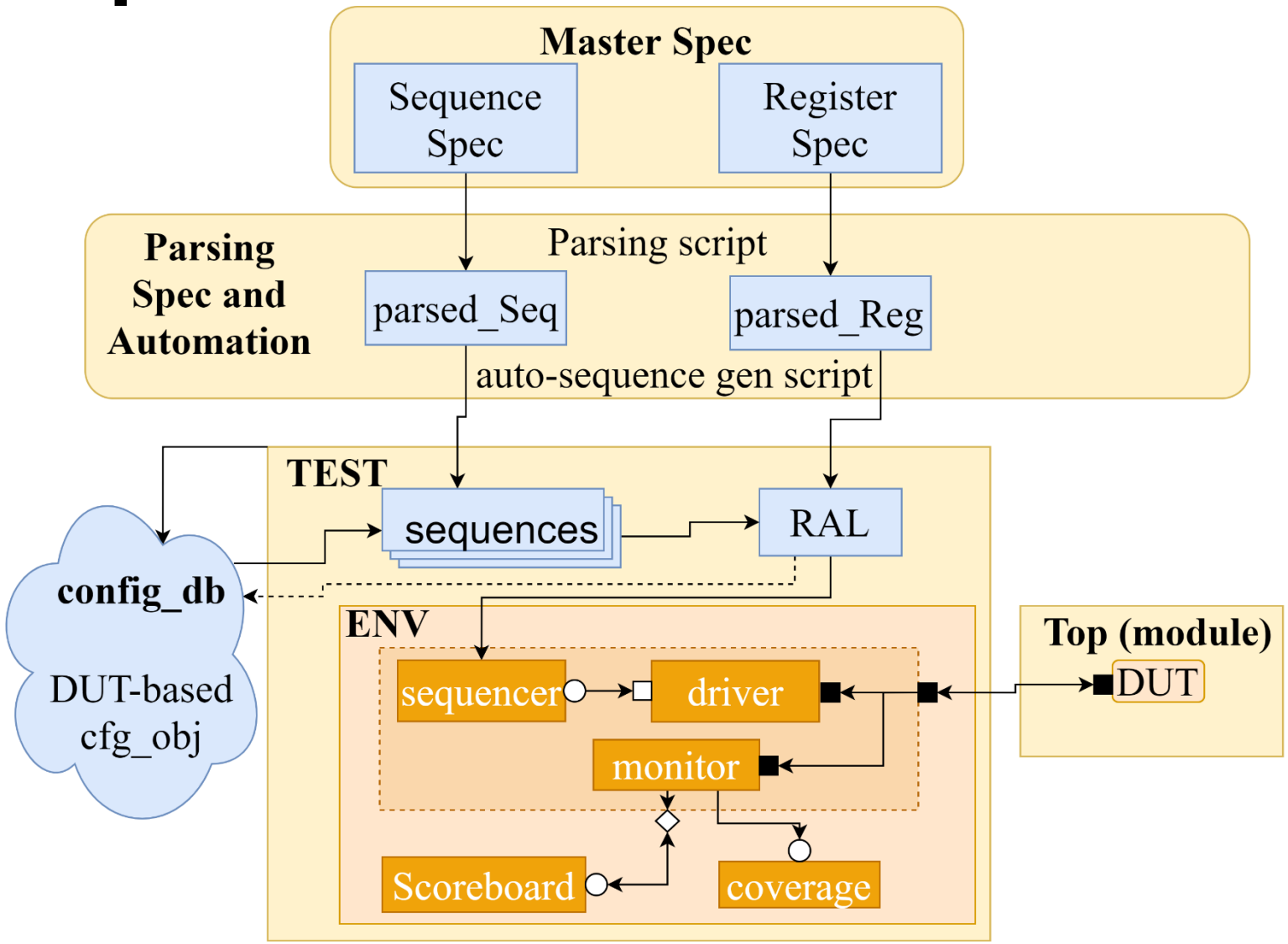
Proposed Automated Val Model

- Solution
  1. RAL and UVM sequences are auto-generated
  2. Define DUT-based configuration objects with links to the RAL



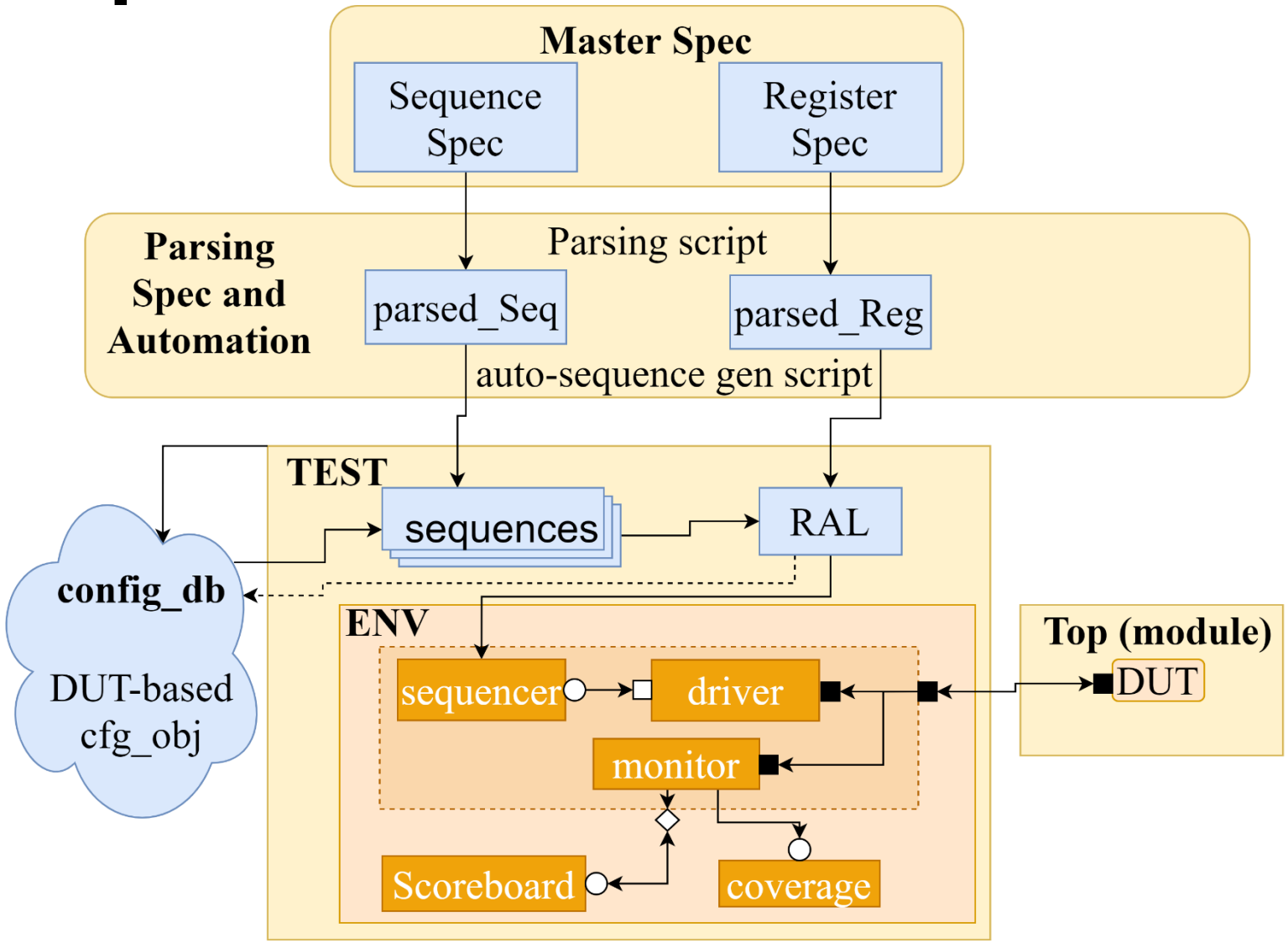
RAL, UVM Sequence Automation

- RAL and UVM sequences are auto-generated
  - "Parsing Spec and Automation" Layer is added



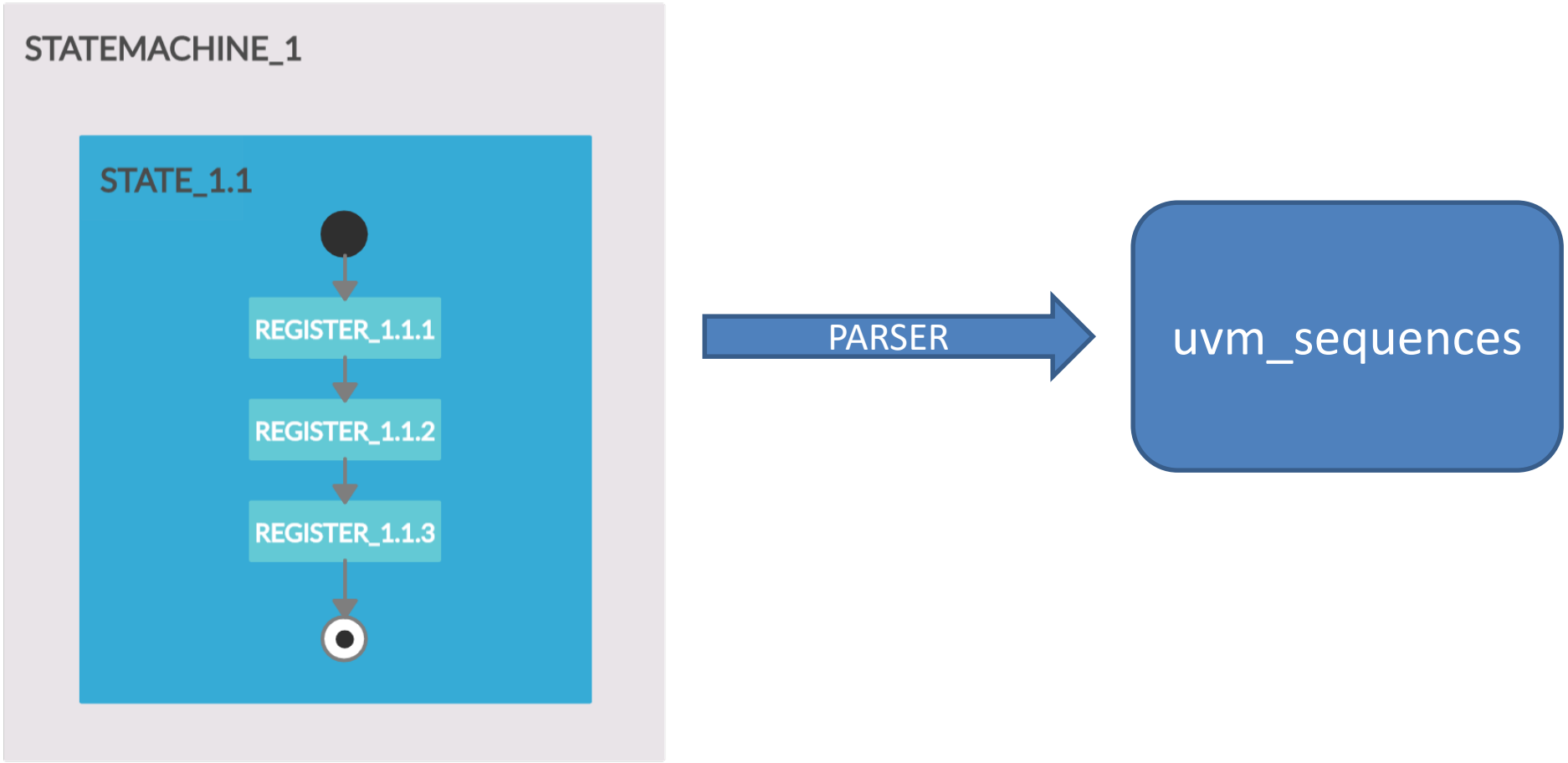
RAL, UVM Sequence Automation

- RAL and UVM sequences are auto-generated
  - A Sequence spec is added alongside the existing Register spec
    - A sequence spec dictates the register programming flow
    - Written in a machine readable State Machine xml format



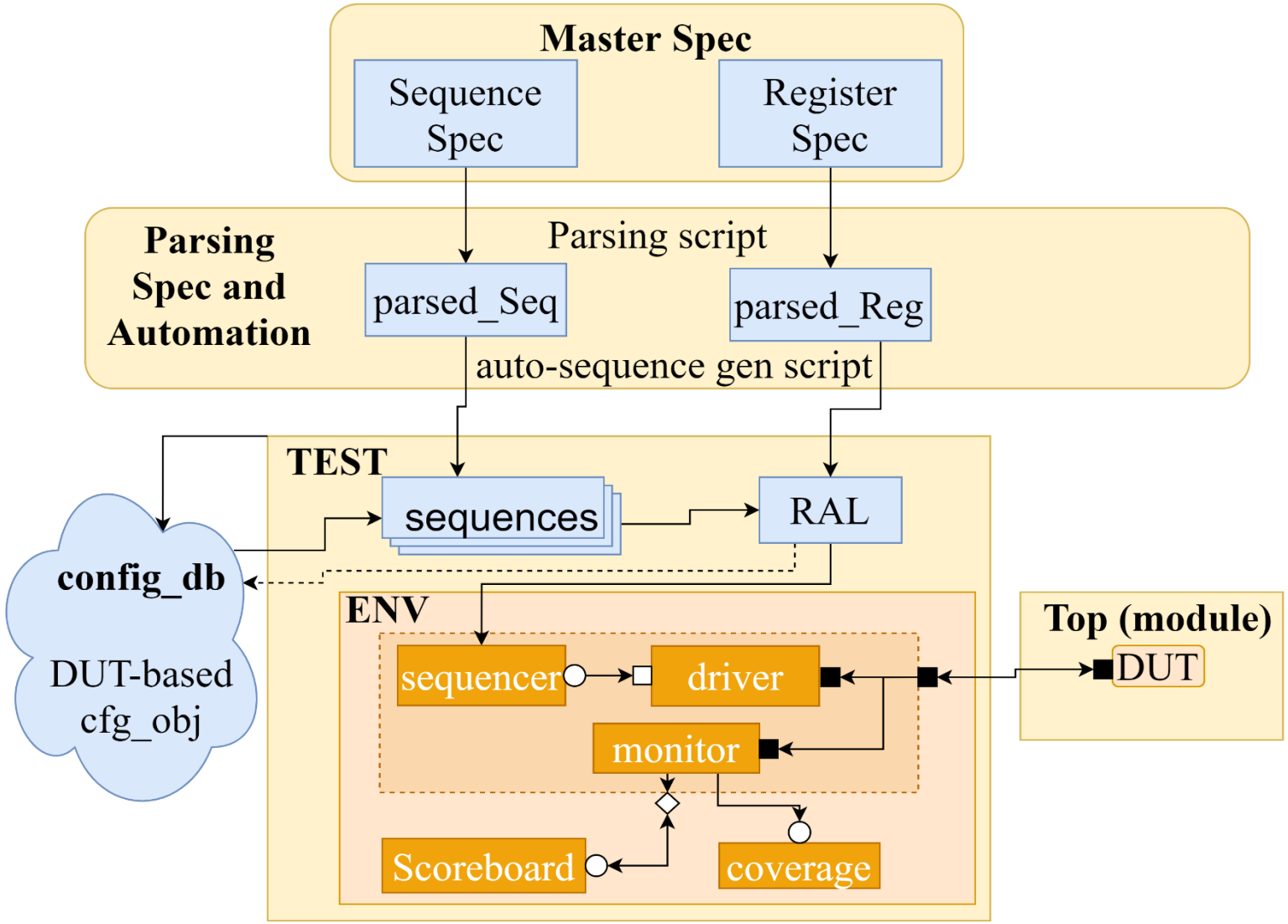
Sequence spec to uvm_seq

- Sequence spec (State Machine DAGs) → uvm_sequences



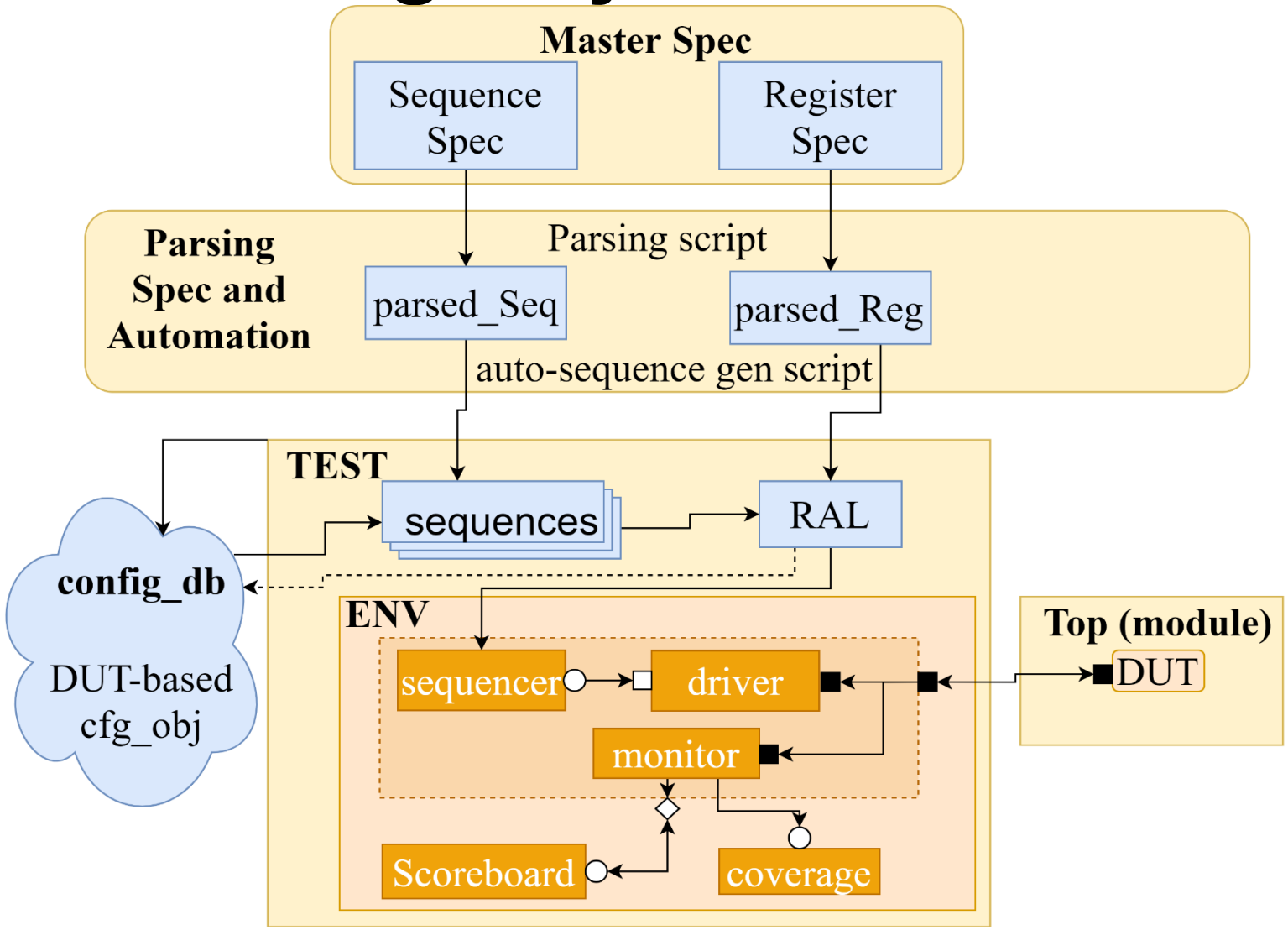
RAL-linked config objects

- Define DUT-based configuration objects
  - DUT hierarchy is determined through these config_objects



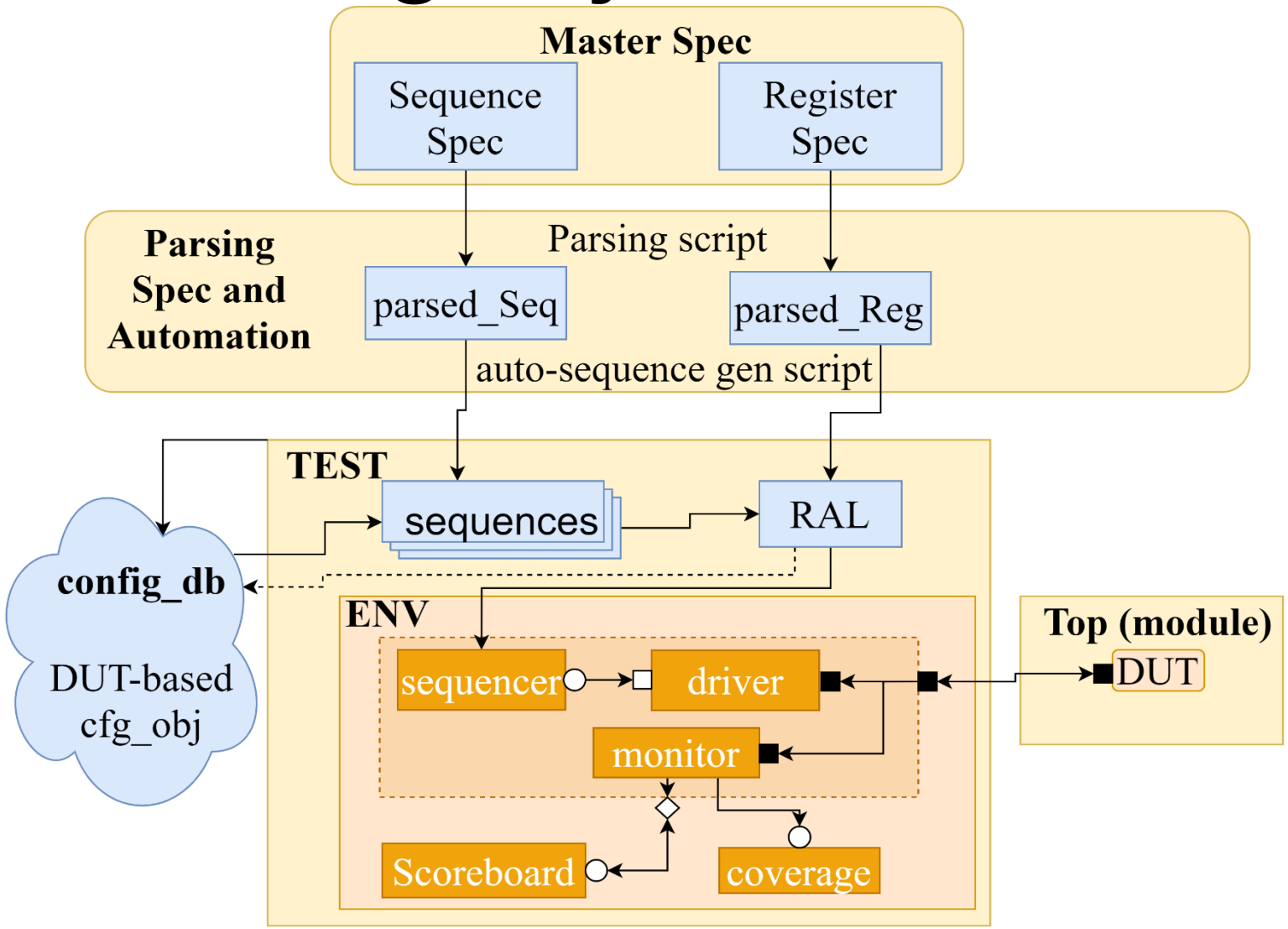
RAL-linked config objects

- Define DUT-based configuration objects
  - Control knobs for DUT specific sequences are defined.



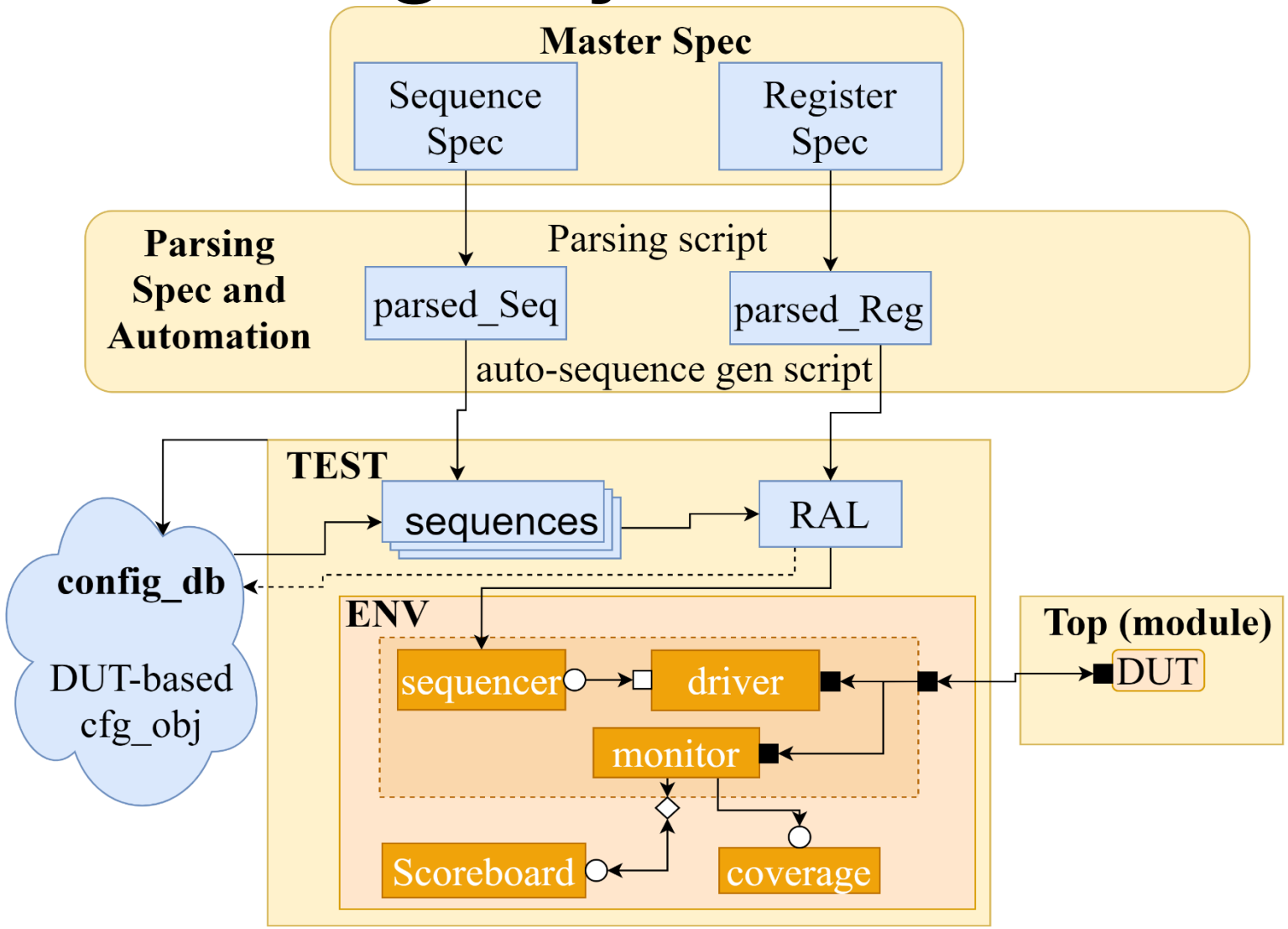
RAL-linked config objects

- Define DUT-based configuration objects
  - Define complex constraints
  - DUT-specific constraints
  - Interdependent constraint resolving.
  - Control knob constraints



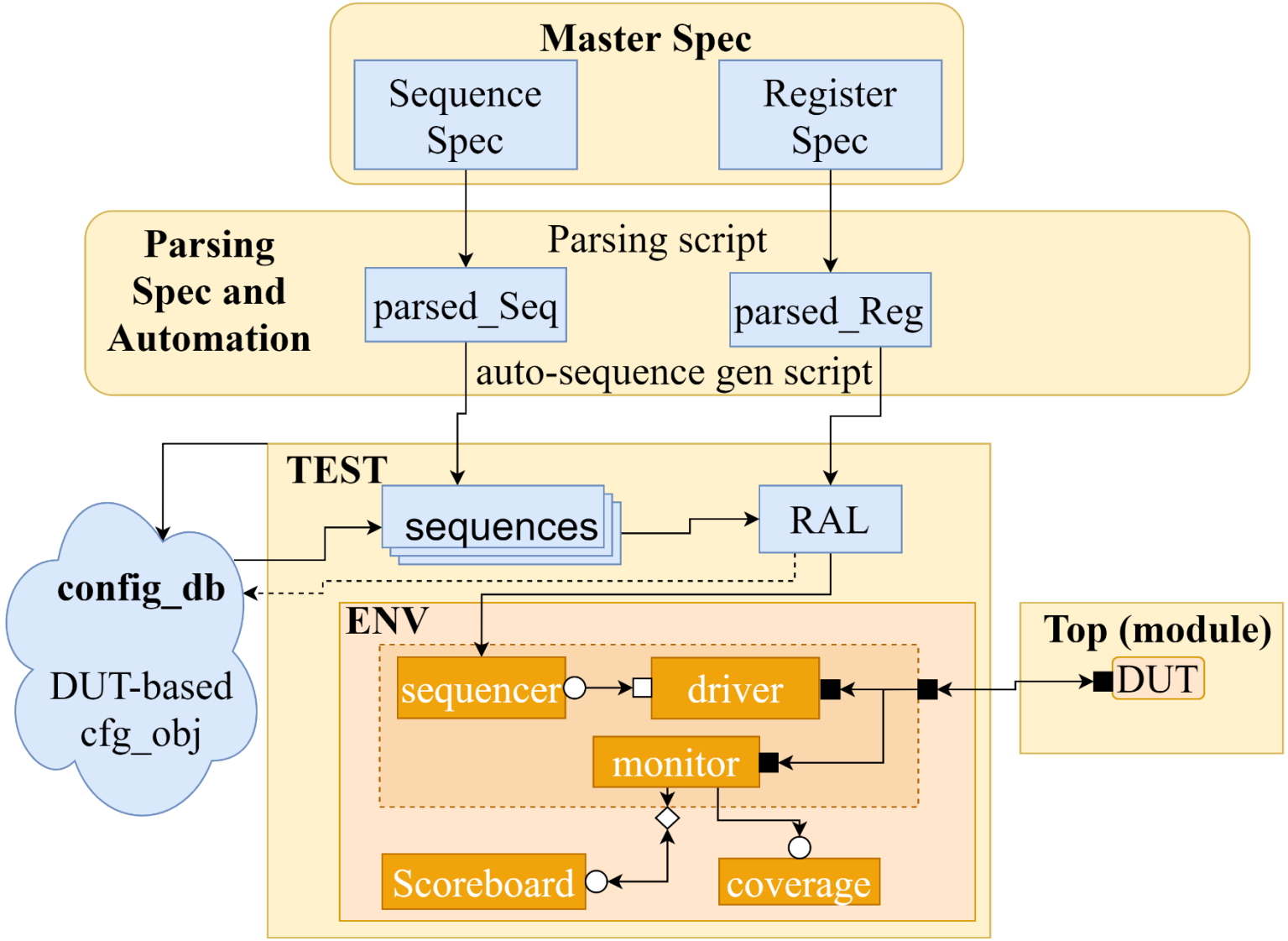
RAL-linked config objects

- Define DUT-based configuration objects
  - Define *rand* pointers to registers used within the DUT
    - Selective RAL randomization is achieved saving sim time



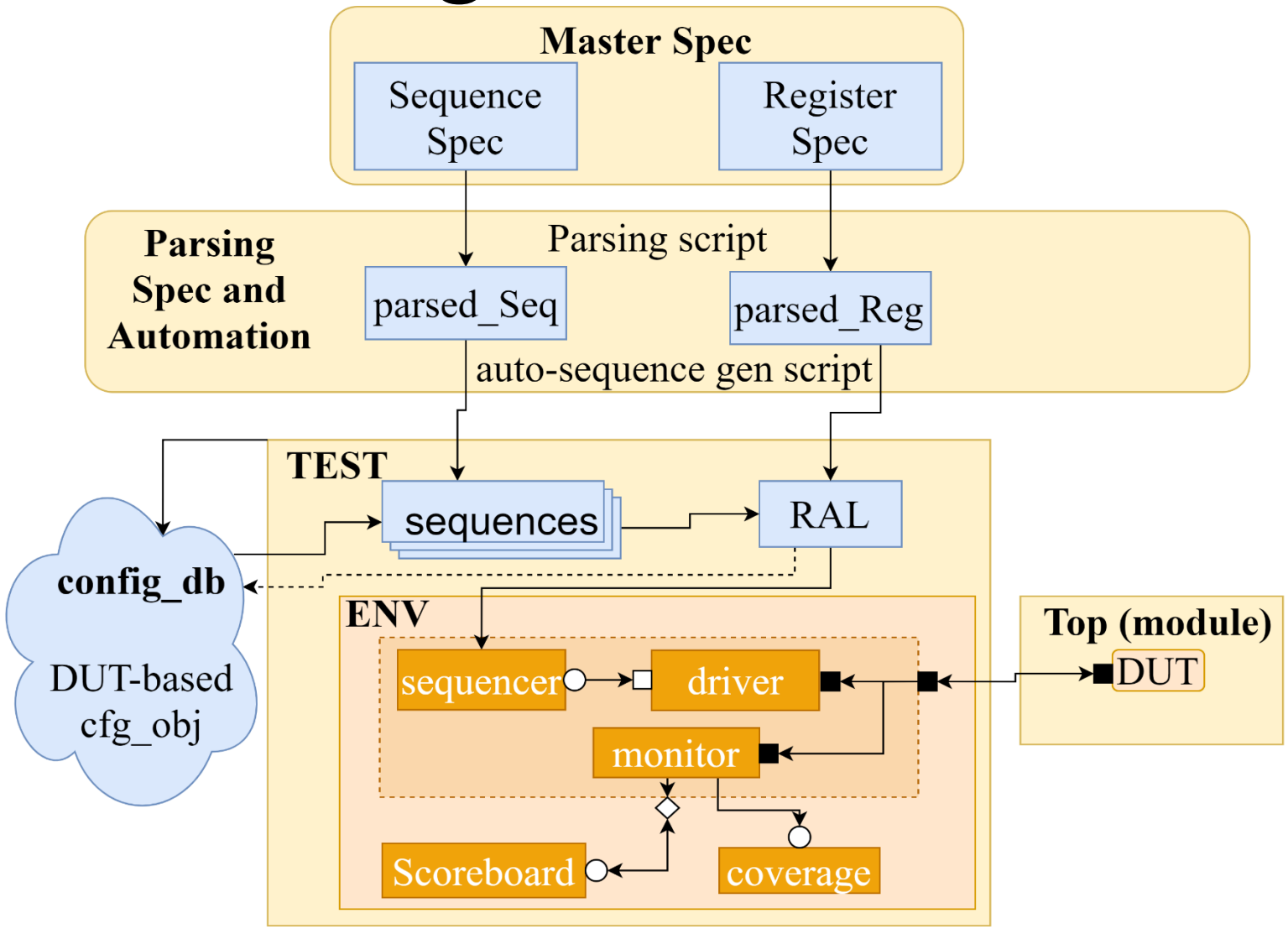
Additional benefits

- Additional benefits
  - Easier test writing
  - Better control over DUT sequences



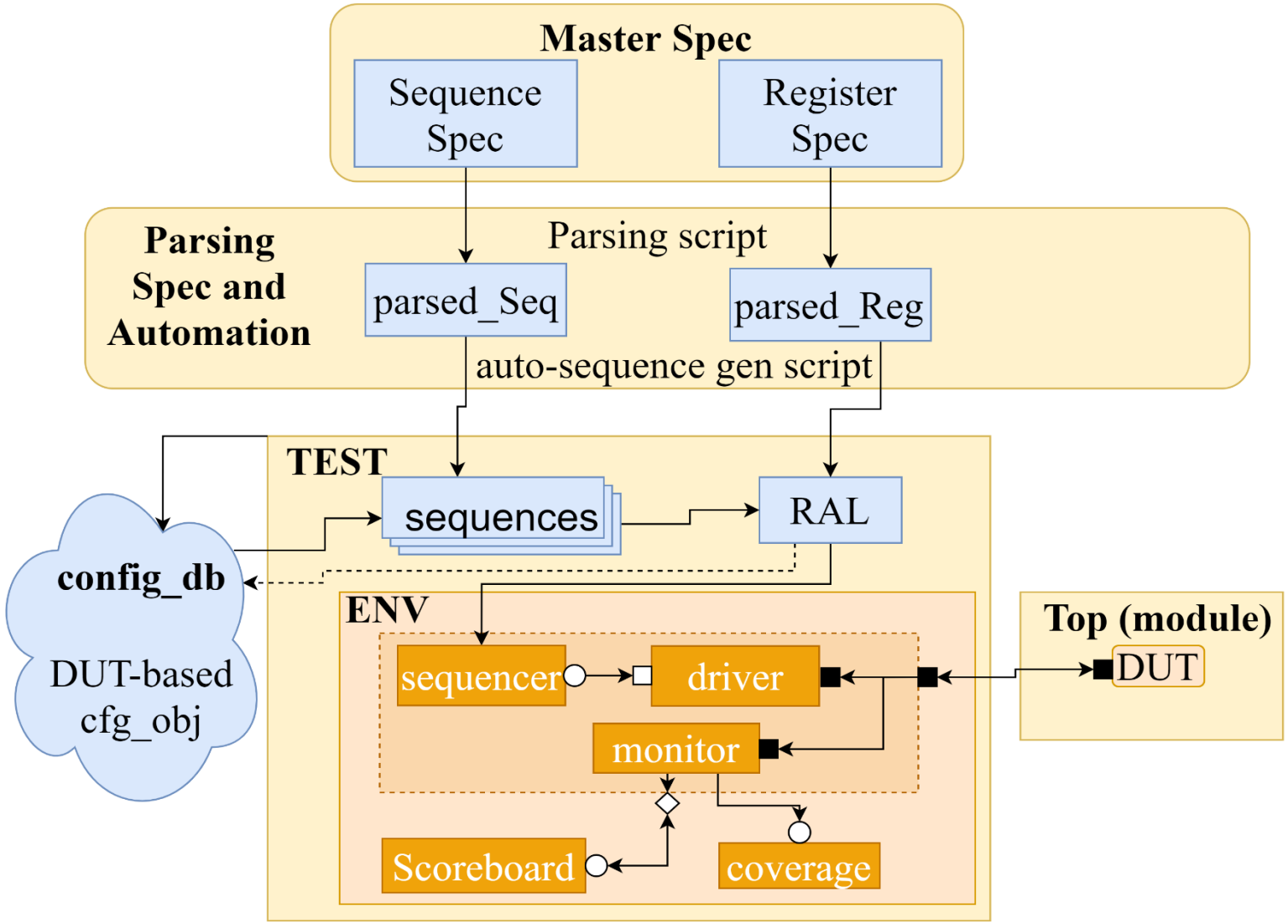
Test Writing

- Uvm_sequences generated in a hierarchical fashion.
- Nested sequences are controlled using control knobs



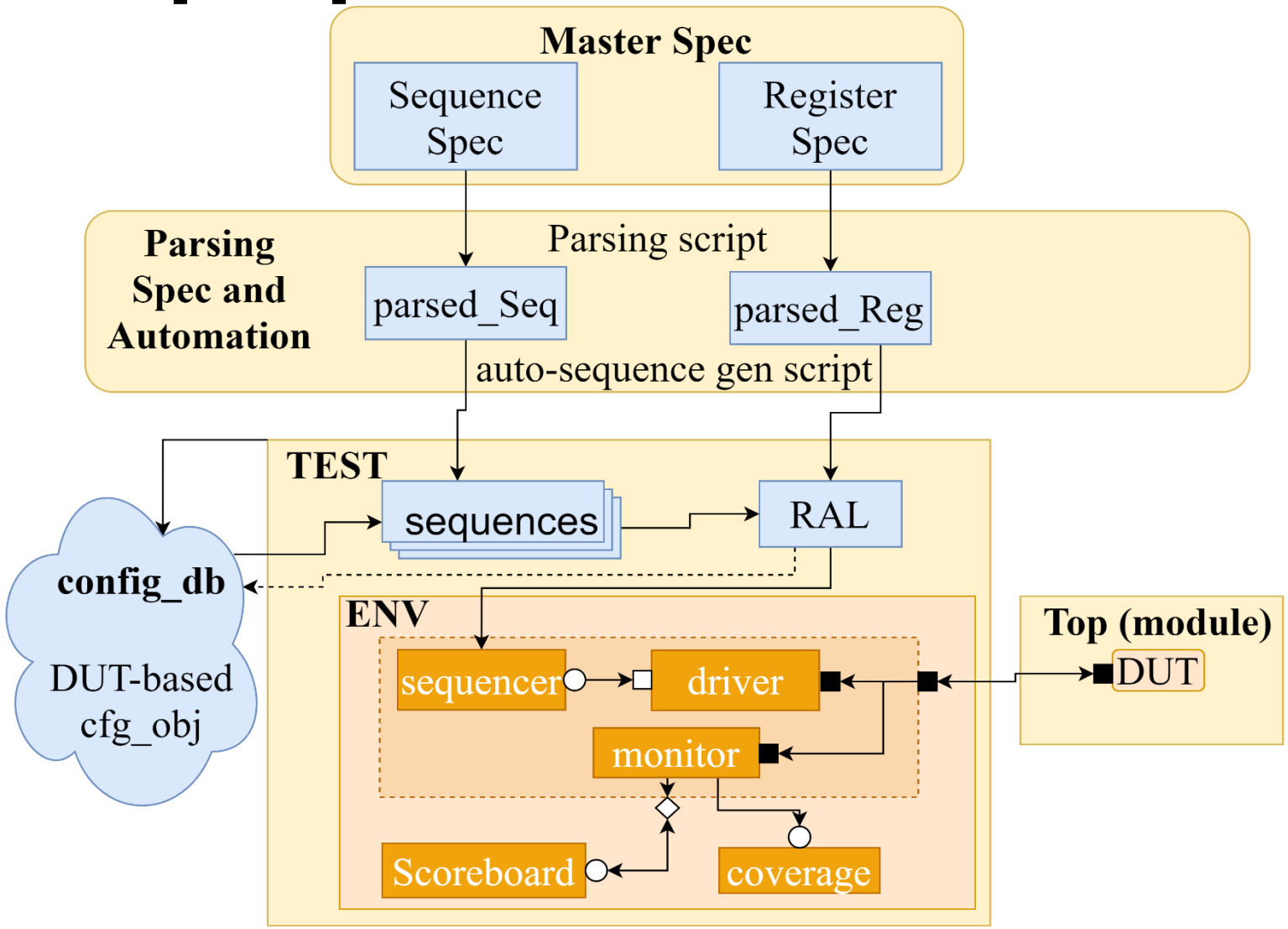
Test Writing

- Control knobs are defined within the DUT-based config objects
- Only start the top virtual sequence on a sequencer and use knobs to control child sequences

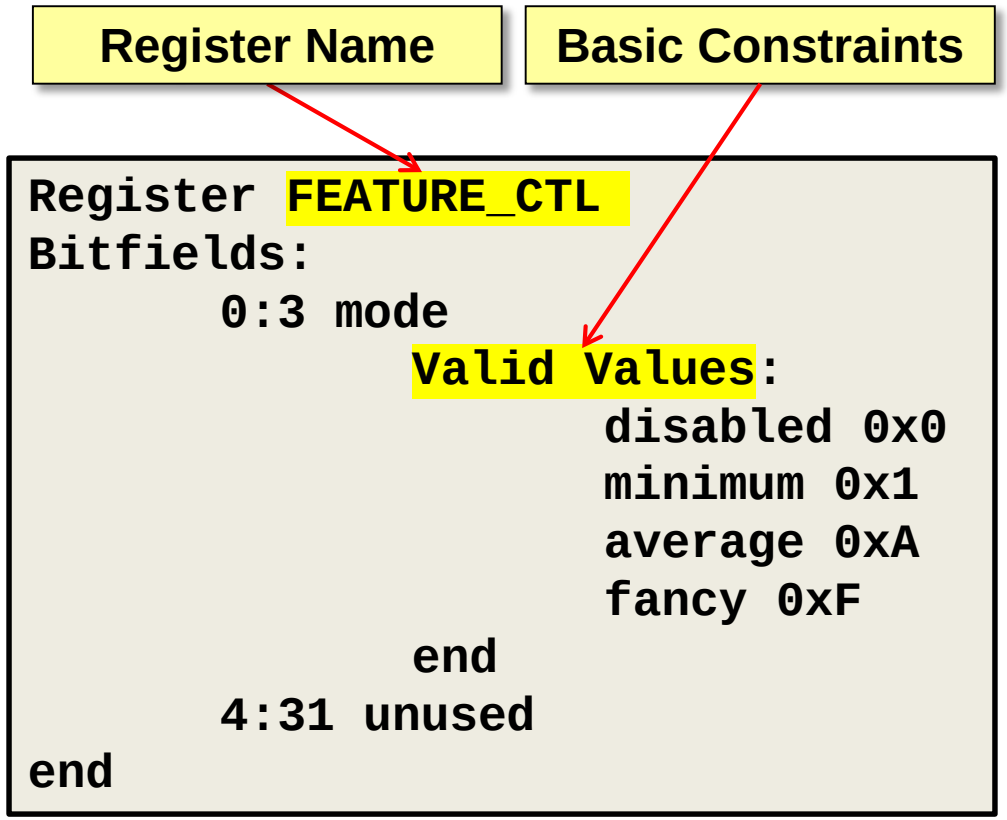


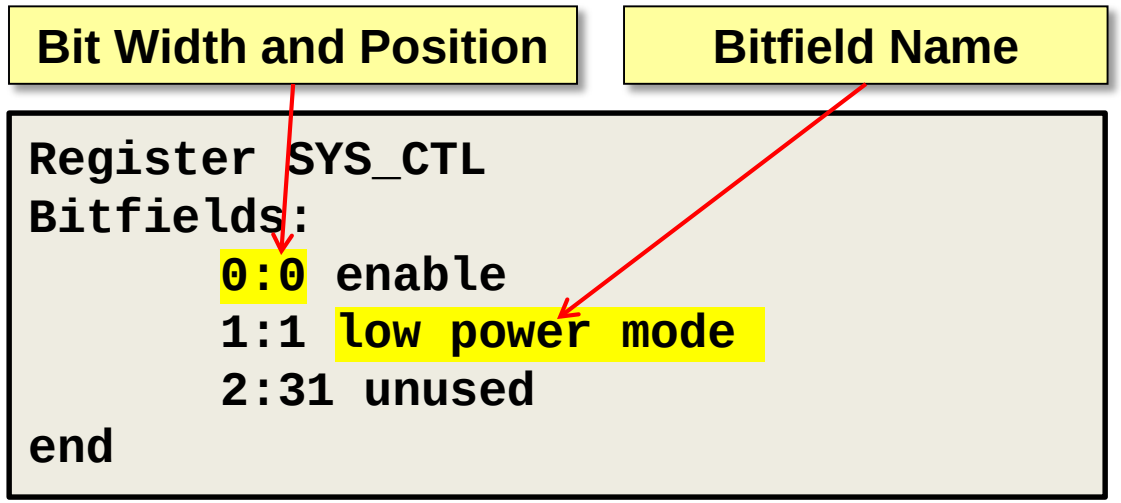
Cons with the proposed Model

- Manually defined config objects
  - Manually updating RAL handles
  - Manually updating constraints



Example Register Spec





Generated Register Class Output

```
class feature_ctl_reg extends uvm_reg;
   rand uvm_reg_field mode;
   uvm_reg_field      unused;
   `uvm_object_utils(feature_ctl_reg)

   // Simple constraint
   constraint feature_ctl_possible_values {
      mode.value inside { 4'h0, 4'h3, 4'hA, 4'hF };
   }
endclass : feature_ctl_reg
```

Autogenerated Output from spec parsing script. Instances of these classes will be members of the RAL class.

```
class system_ctl_reg extends uvm_reg;
    rand uvm_reg_field enable;
    rand uvm_reg_field low_power_mode;
    uvm_reg_field      unused;
    `uvm_object_utils(system_ctl_reg)
endclass : system_ctl_reg
```

Example Sequence Spec

- Very basic XML style spec
- Easy to write
- Can add other features, just requires parser support:
  - Value constraints
  - Task order randomization
  - Subsequence support

```
<Sequence Name="FEATURE_ENABLE">
    <Task Name="Enable_System">
        <Write Register="SYS_CTL"/>
    </Task>
    <Task Name="Enable_Feature">
        <Write Register="FEATURE_CTL"/>
    </Task>
</Sequence>
```

Example Generated Sequence

- Generated from spec parsing script.
- Sequences can be called from test
- Virtual sequences may also be autogenerated.

Update call writes the config randomized value to register.

```
class feature_enable_seq extends uvm_sequence;
    …
    task body();
       super.body();
       ral_instance.feature_ctl.update(status);
       if(status != UVM_IS_OK)
              //Report error
       ral_instance.sys_ctl.update(status);
       if(status != UVM_IS_OK)
              //Report error
    endtask
endclass
```

RAL instance acquired from the UVM config database. It was placed there by the test environment.

Example Test

Randomizing config object randomizes the RAL's register values

Starting the sequence sends register writes to the system through the RAL

```
task run()
    cfg.randomize() with {
        //Ensures the feature is enabled
        cfg.feature_ctl.mode.value != 1'b0;
        //Ensures the system is enabled
        cfg.sys_ctl.enable.value == 1'b1;
    };

    ftr_enable_seq.start(env.agent.sequencer);
    …
endtask
```

Example Configuration Object Pt. 1

Contains instances of all of the registers in this unit.

Can be difficult to automatically add cross register constraints from spec. These can be added to the config object.

```
class sys_cfg extends uvm_object;
    rand feature_ctl_reg feature_ctl;
    rand sys_ctl_reg sys_ctl;
    `uvm_object_utils(sys_config)

    constraint low_power_mode_nothing_fancy {
        if(sys_ctl.low_power.value == 1'b1)
            feature_ctl.mode.value != 0xF;
    }
    …
endclass
```

Example Configuration Object Pt. 2

Grabs the RAL from the UVM config db. It was created in the environment

Links the configs registers to the RALs registers

```
task connect();
  ral_class regs;
  if(
     !uvm_config_db#(ral_class)
     ::get(null, "*", "regs", regs)
    )
  begin
    `uvm_error("sys_cfg.connect()","Where's the RAL?")
  end

  this.sys_ctl         = regs.sys_ctl;
  this.feature_ctl     = regs.feature_ctl;
endtask
```

Example Configuration Object Pt. 3

Defining control knobs of hierarchical sequences.

Constraints defined to control specific sequences

```
class top_cfg extends uvm_object;
    rand control_knob top_virtual_seq;
    rand control_knob child1_seq;
    rand control_knob child2_seq;
    `uvm_object_utils(sys_config)

    constraint mutually_exclusive_sequences {
        if(child1_seq.signal == 1'b1)
            child2_seq.signal == 1'b0;
    }
    …
endclass
```

Example Test

Controlling child sequences

Starting the top sequence

```
task run()
    cfg.randomize() with {
        //Sets the control knob for child sequence
        cfg.child1_seq == 1'b1;
    };

    top_virtual_seq.start(env.agent.sequencer);
    …
endtask
```

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Questions?

accellera SYSTEMS INITIATIVE