A Comprehensive UVM Verification Environment for MPEG Transport-Stream Processing

Filippo Borlenghi

ALi Europe

Plan-les-Ouates, Switzerland

June 18th, 2015

SNUG Grenoble

Agenda

Target application and verification goals

UVM testbench structure

Configuration mechanism

Tests

Assertions and coverage

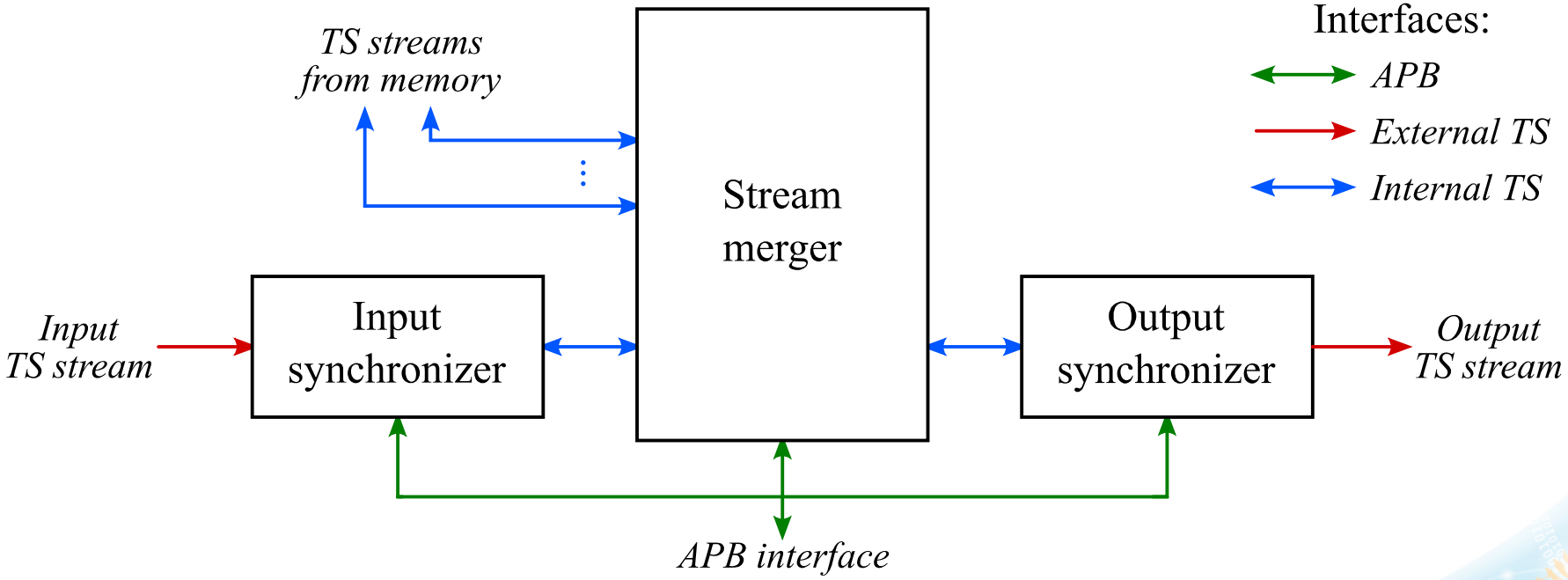
Conclusions

Target Application

Design Under Verification

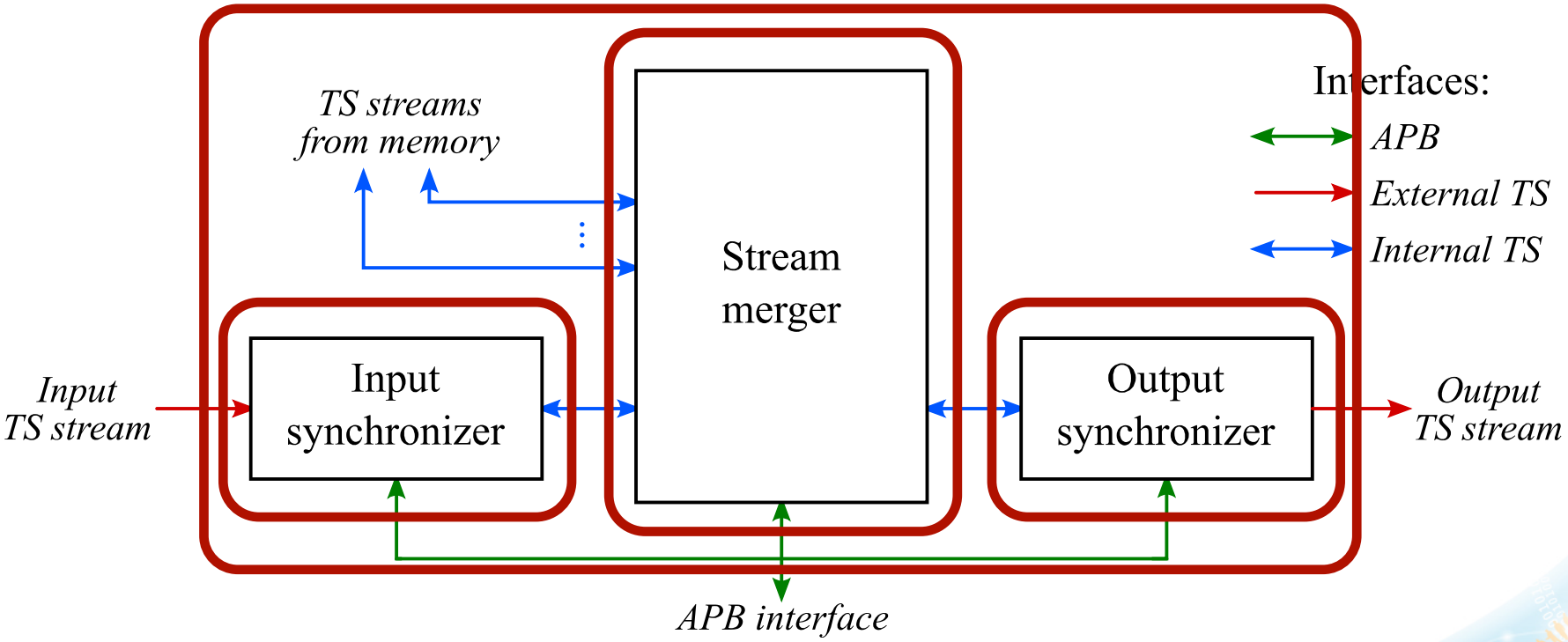
MPEG transport-stream (TS) processing

- Inputs: one external TS stream + N TS streams from memory
- Output: merged TS stream
- Merge policy: round-robin or timing-aware



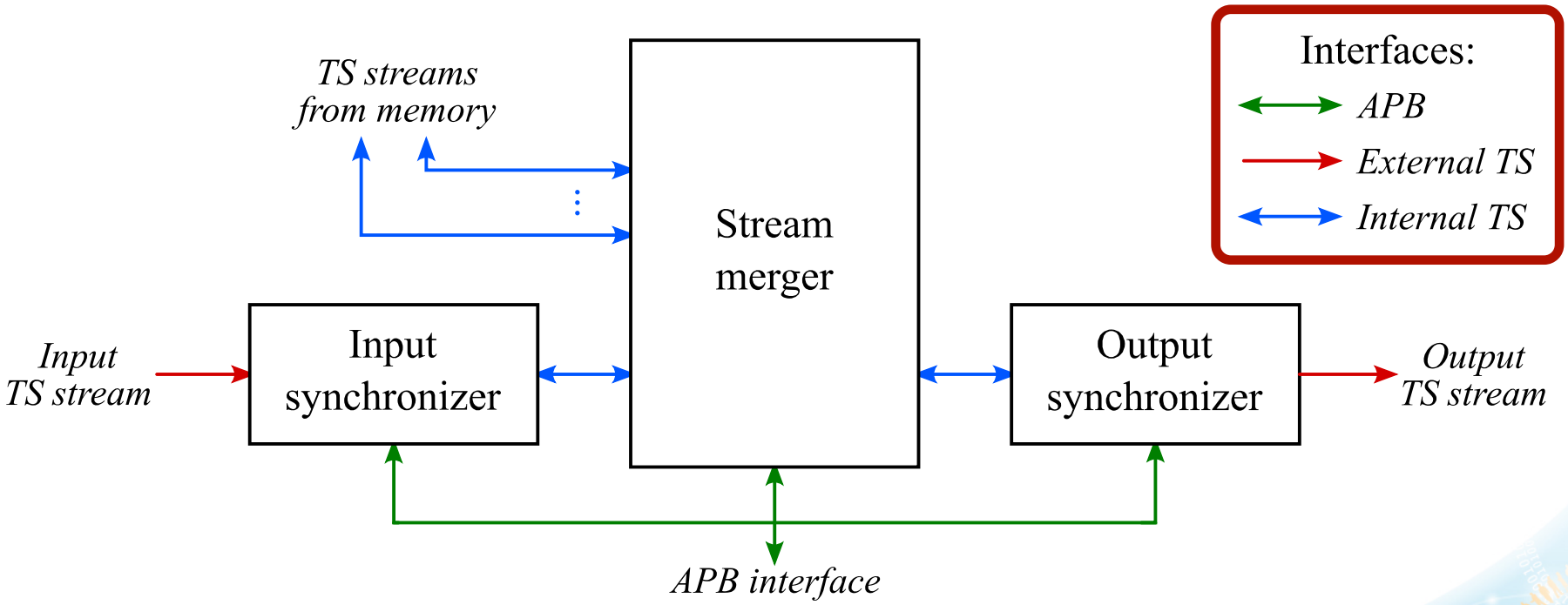
Verification Goals

- Single blocks + system functionality



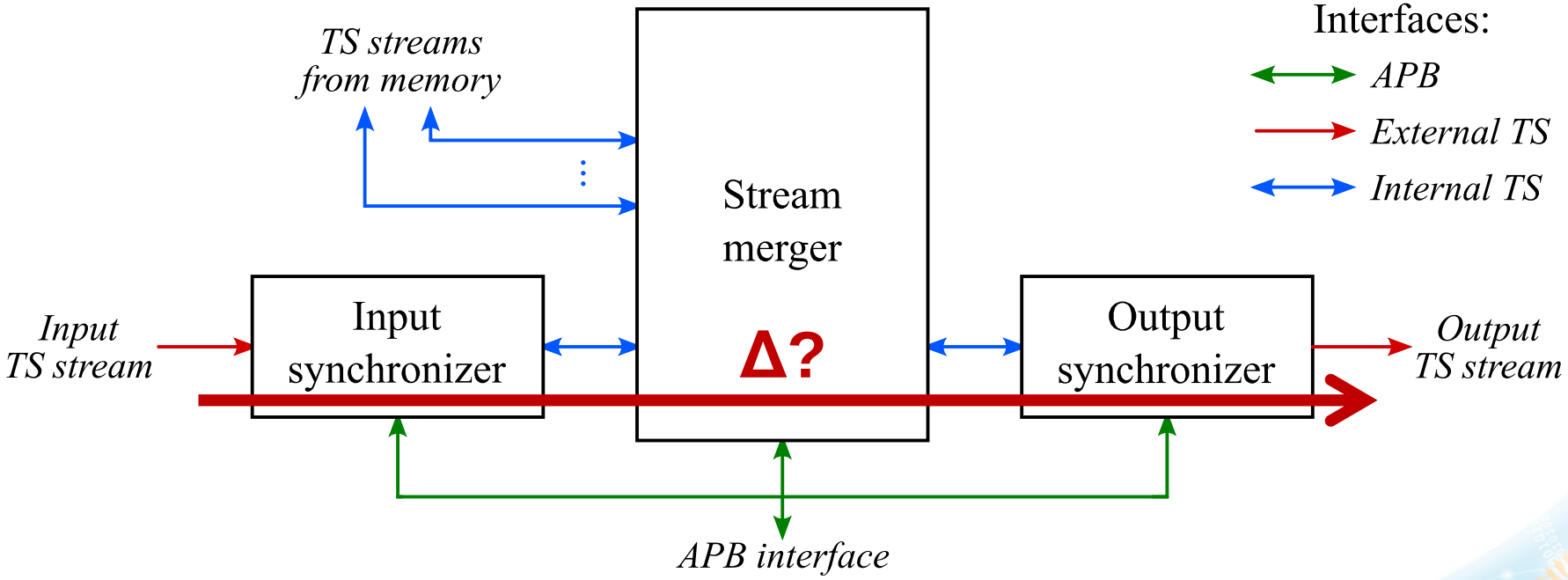
Verification Goals

- Single blocks + system functionality
- Interfaces



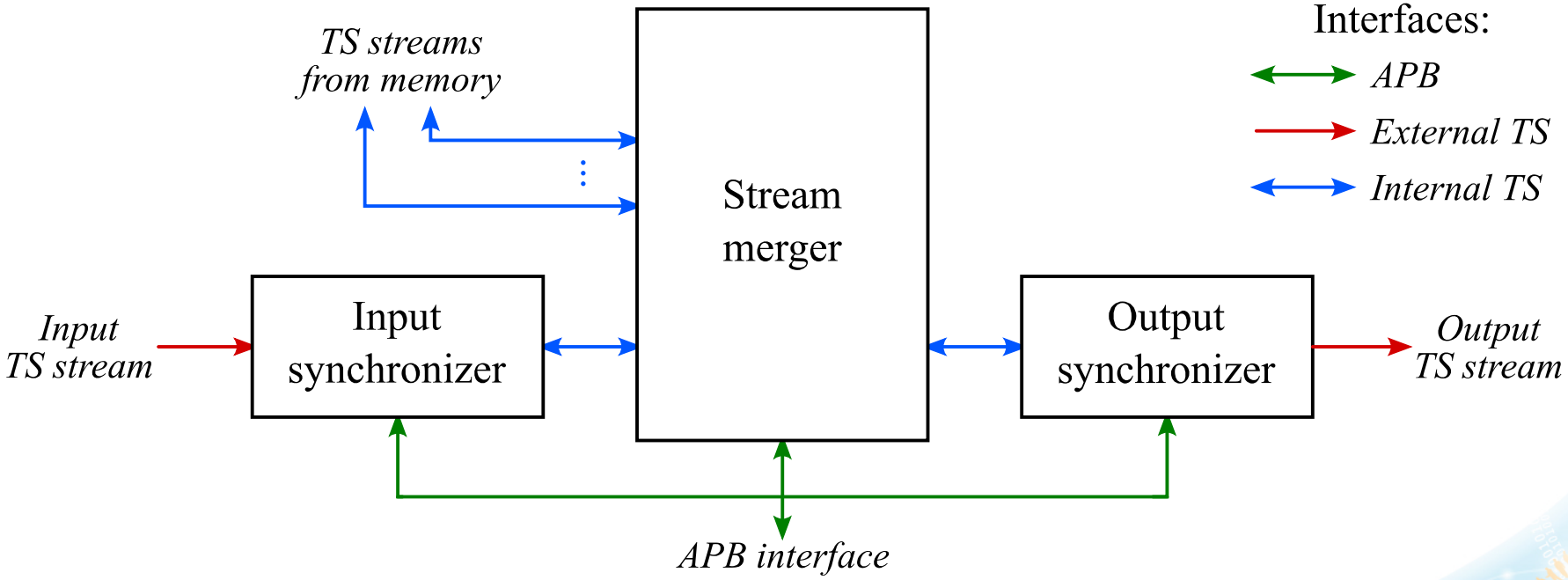
Verification Goals

- Single blocks + system functionality
- Interfaces
- Timing properties (latency, throughput)



Verification Goals

- Single blocks + system functionality
- Interfaces
- Timing properties (latency, throughput)
- Reconfigurability

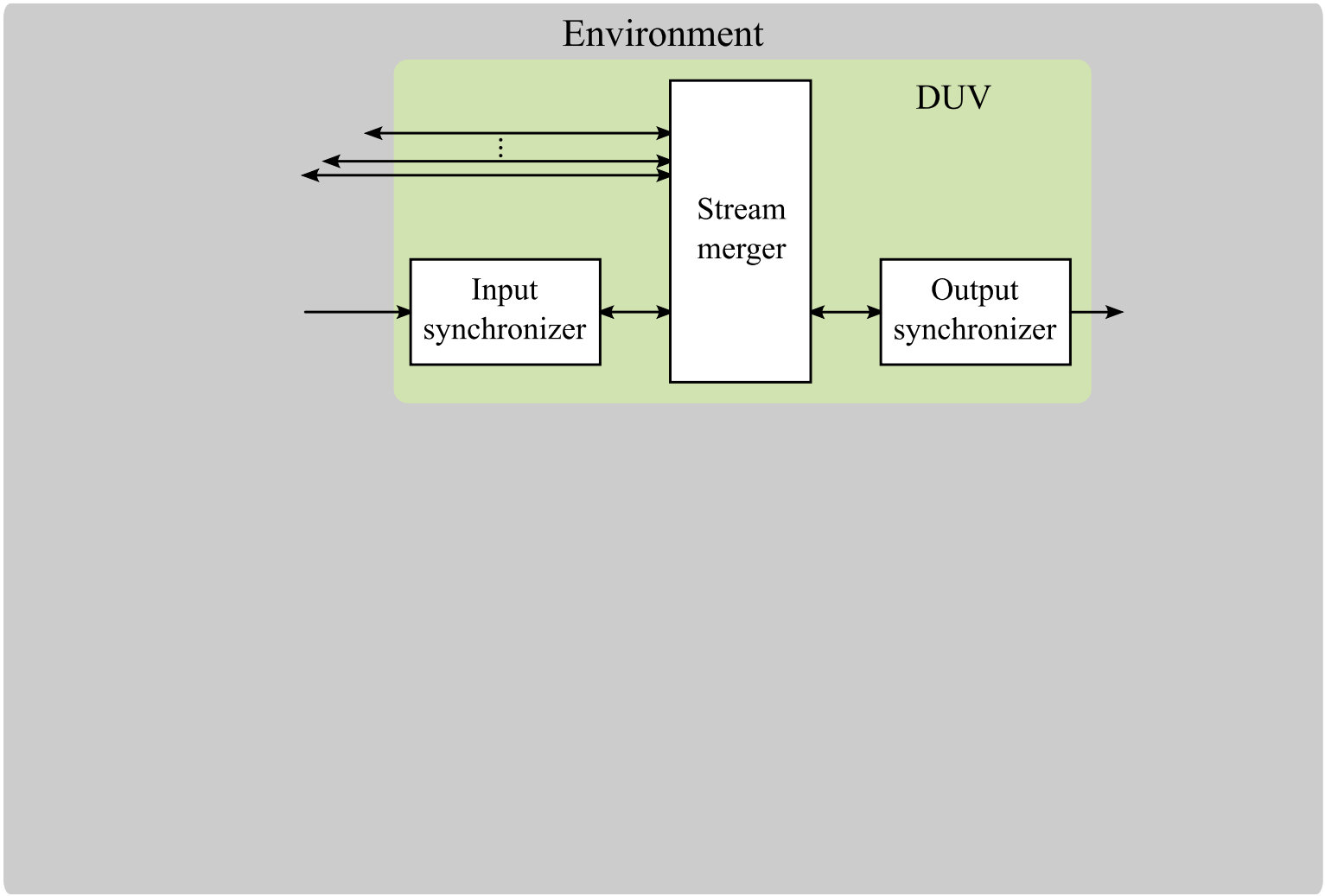


Why UVM?

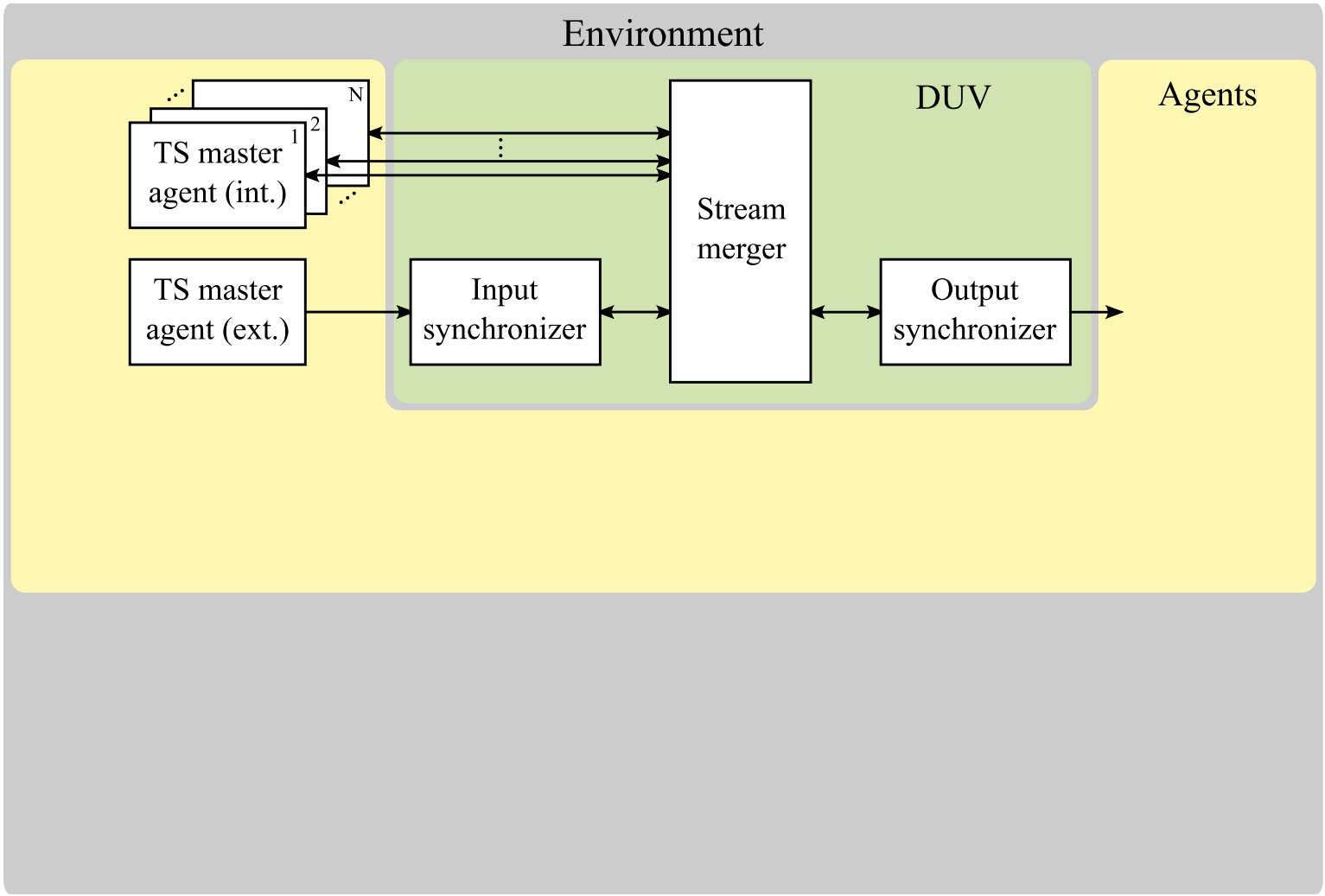
- Object oriented
  - → Modular
  - → Reusable

  } ≠ Monolithic SystemVerilog tb. (previous approach)
  - → Parameterizable
- Register abstraction layer
  - → Simplified register access
- SystemVerilog advanced verification features
  - → Constrained random verification
  - → Assertions
  - → Coverage

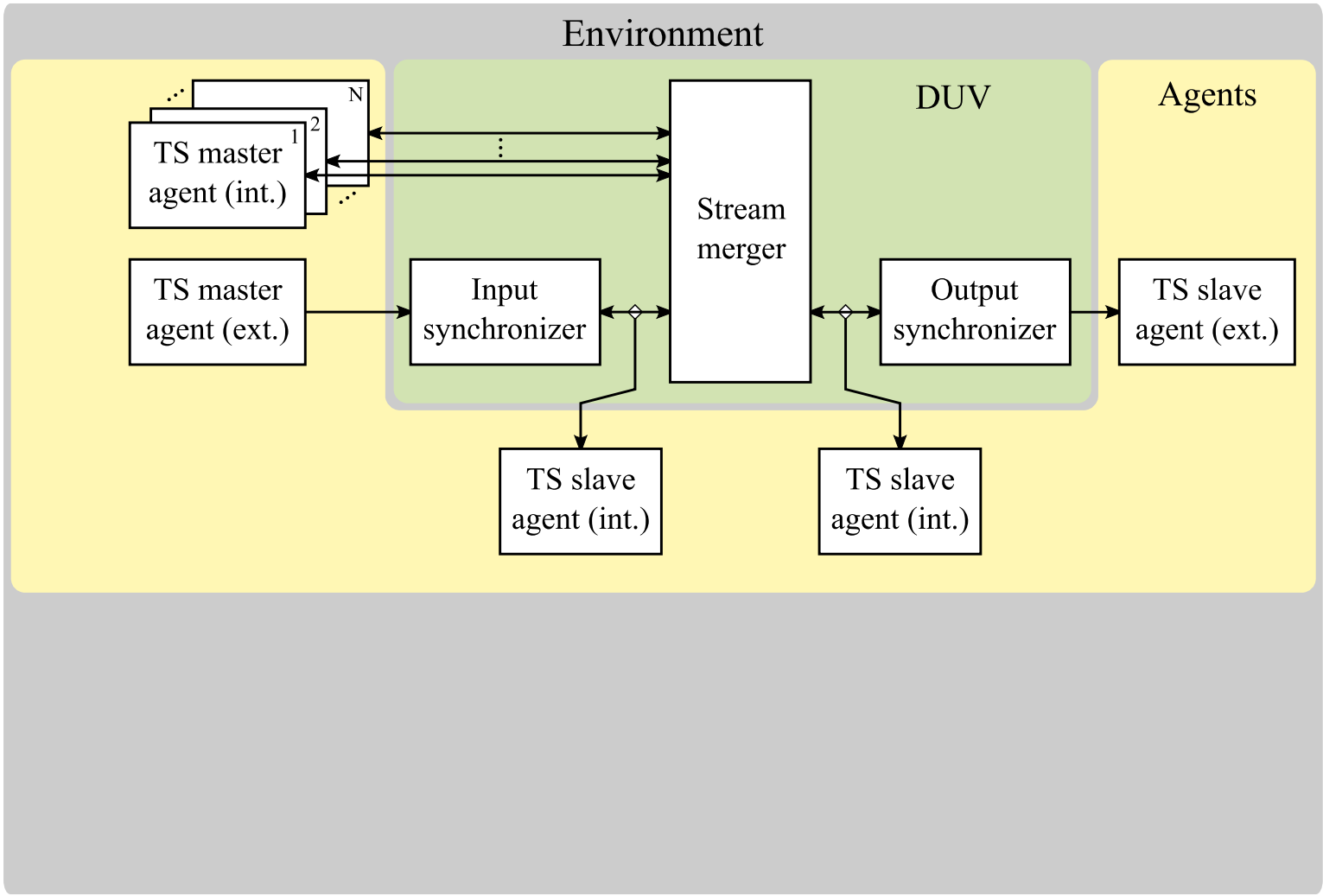
Testbench Structure



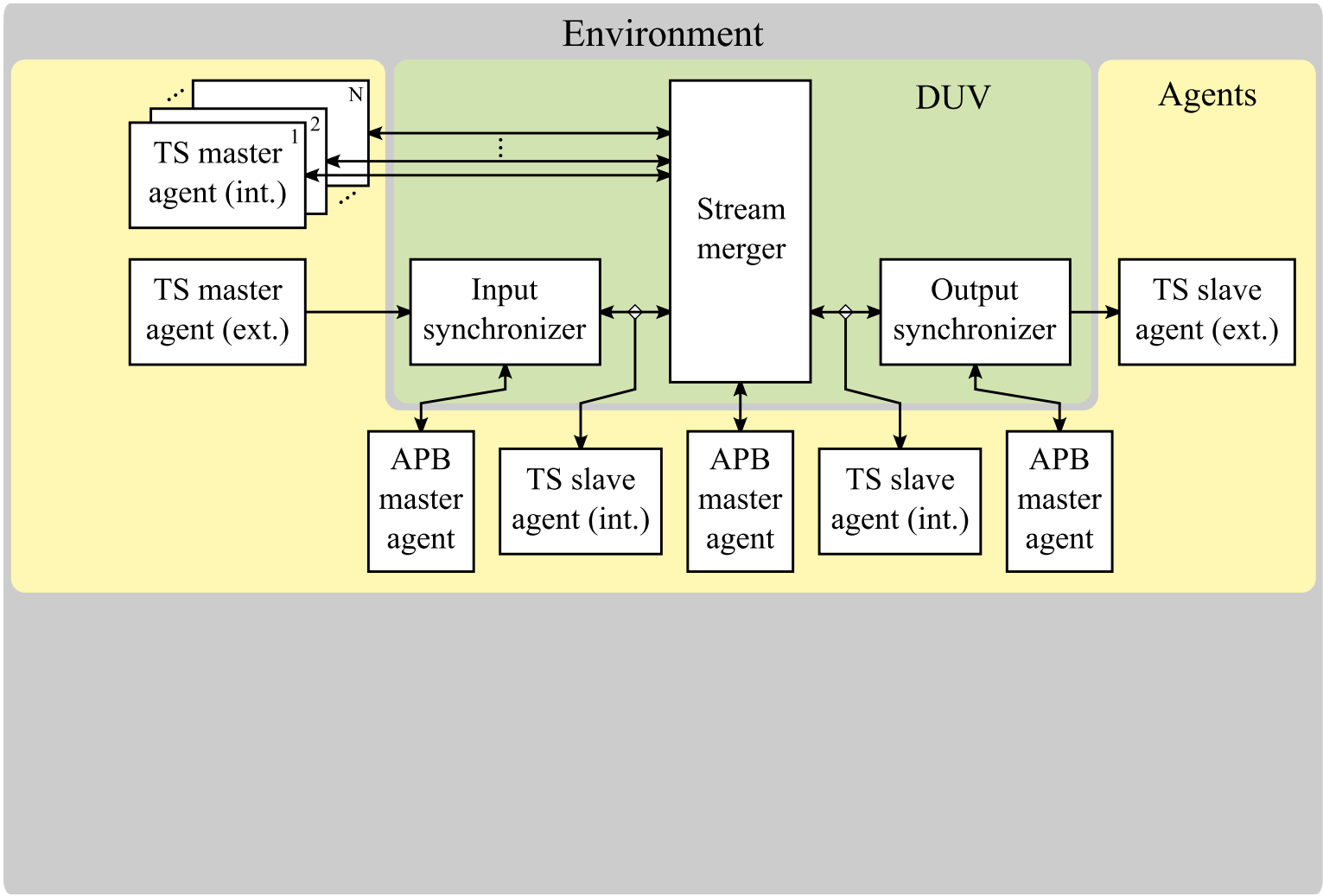
Testbench Structure



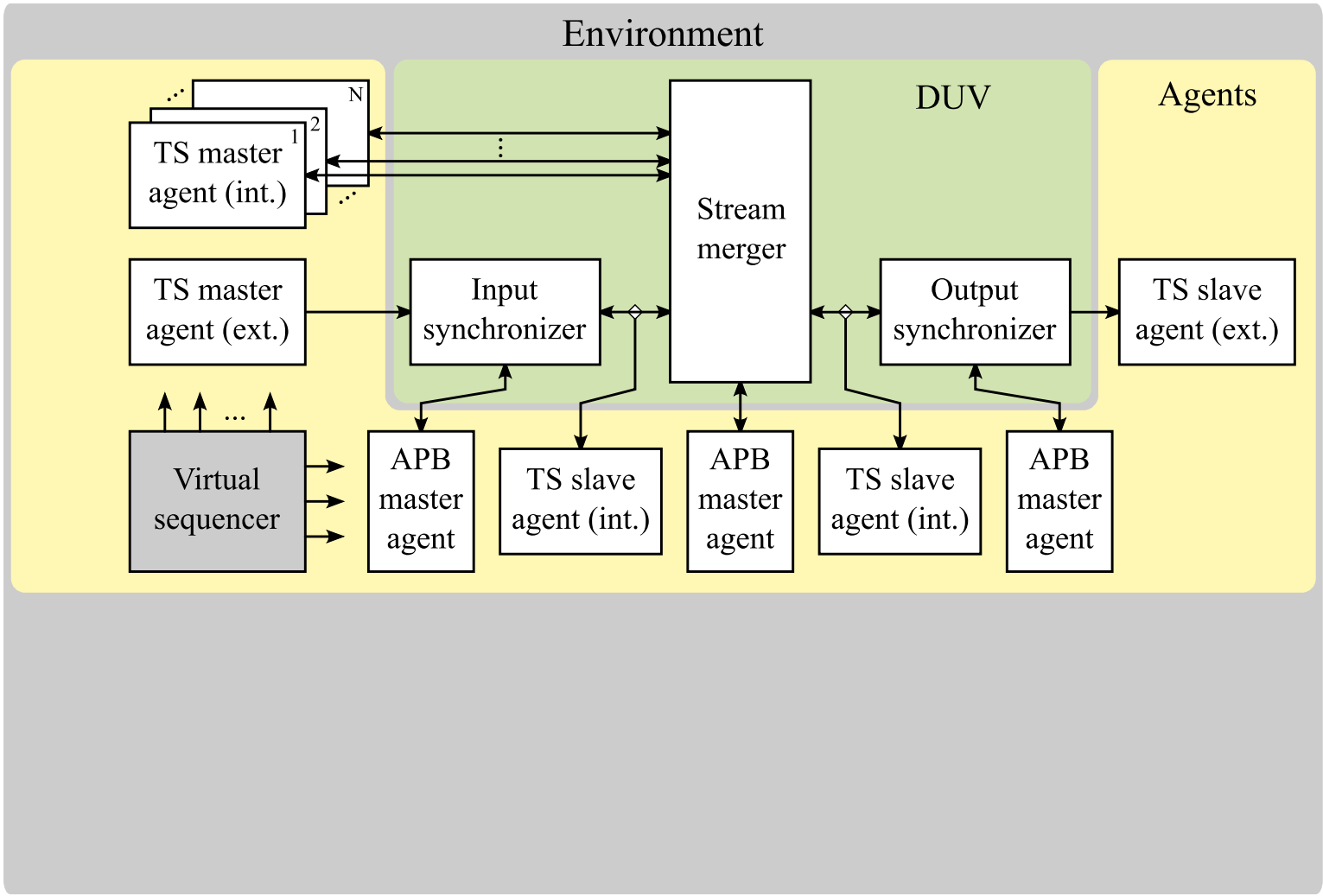
Testbench Structure



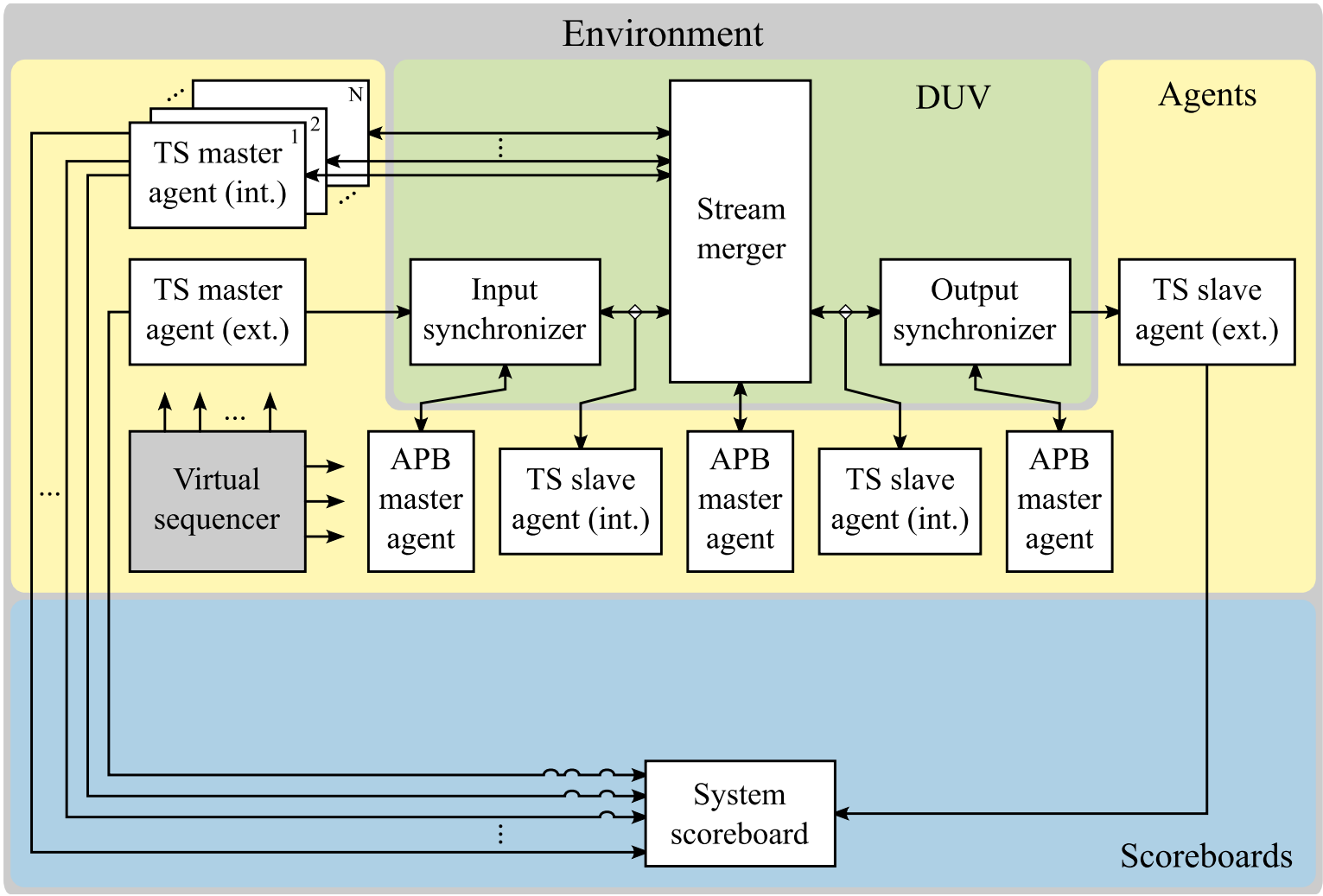
Testbench Structure



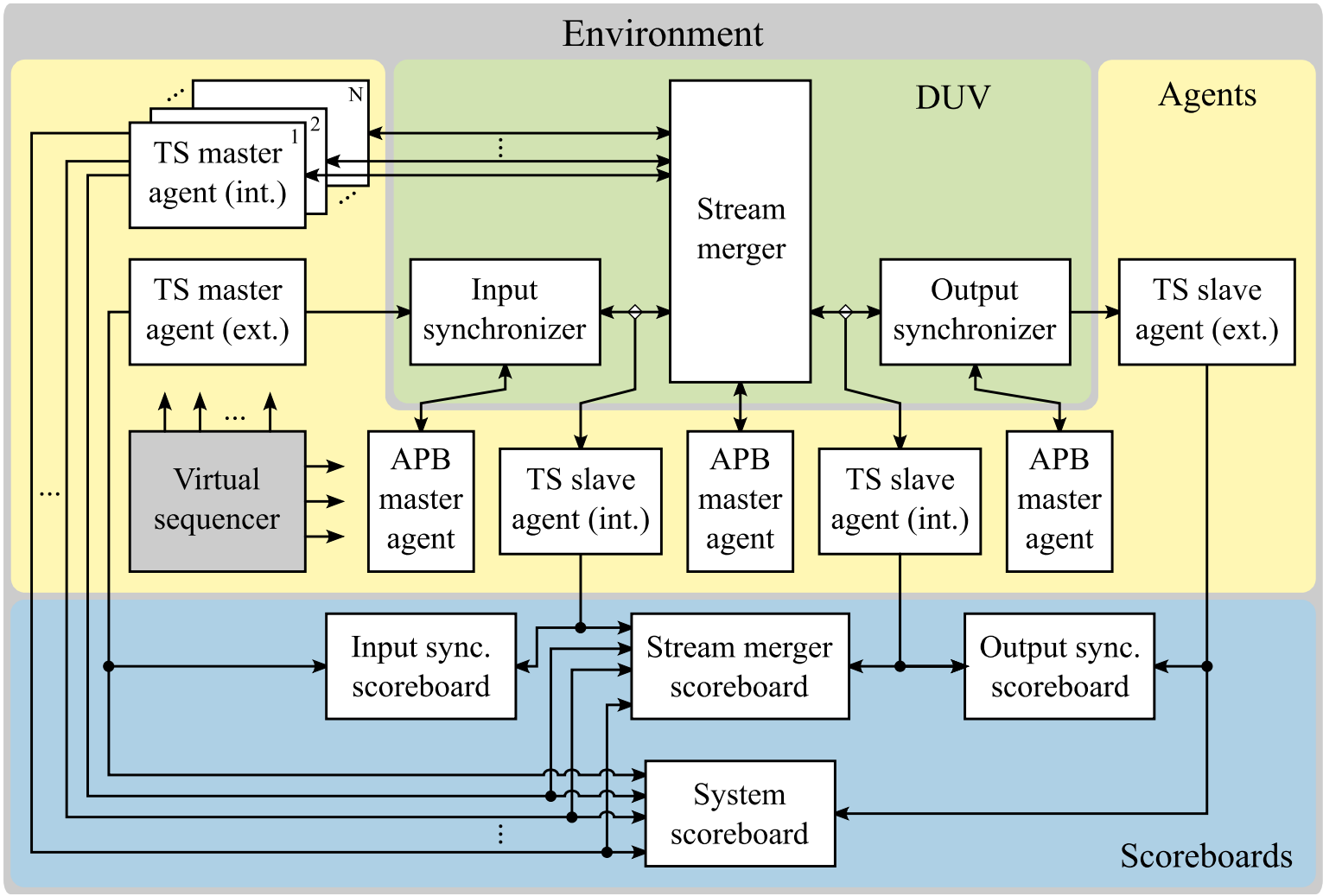
Testbench Structure



Testbench Structure



Testbench Structure



Hierarchical Approach

Set of classes for each block (independent of top level)

- High-level configuration
- RAL register model
- APB configuration sequences
- Transformer (golden model)
- Block-level scoreboard

+

Top-level components

- Virtual sequencer
- System scoreboard

Configuration Mechanism

Synopsys Users Group

RAL model | Config. class | Config. library | Config. sequence

```
block OSYNC {
  register CFG (i_osync_regif.cfg_reg) @'h0000 {
    # select internal or external clock, rising or falling edge
    field CLK_EDGE @2 { enum {FALL, RISE}; bits 1; reset 1'h0; access rw; }
    field CLK_SEL  @1 { enum {EXT,  INT }; bits 1; reset 1'h0; access rw; }
    field OSYNC_EN @0 { enum {DIS,  ENA }; bits 1; reset 1'h0; access rw; }
  }
  register INTCLK (i_osync_regif.intclk_reg) @'h0004 {
    # period of internally-generated clock
    field TCLK @0 { bits 24; reset 24'h000000; access rw; }
  }
}
system osync_regmodel {
  block OSYNC (i_osync) @'h0000;
}
```

Configuration Mechanism

Synopsys Users Group

RAL model | **Config. class** | Config. library | Config. sequence

```
class osync_config extends uvm_object;

  typedef bit [`APB_DATA_WIDTH-1:0] apb_data_t;
  rand bit osync_en; rand bit clk_sel; rand bit clk_edge;
  rand int intclk_frequency; // in MHz

  function apb_data_t get_cfg();
    apb_data_t tmp_reg = 'b0;
    return (tmp_reg | (cfg_osync_en<<0 | cfg_clk_sel<<1 | cfg_clk_edge<<2));
  endfunction : get_cfg

  function apb_data_t get_intclk();
    apb_data_t tmp_reg = 'b0;
    bit[23:0] intclk_tclk = real'(`SCALING_FACTOR) / real'(intclk_frequency);
    return (tmp_reg | intclk_tclk);
  endfunction : get_intclk

endclass : osync_config
```

Configuration Mechanism

RAL model | Config. class | **Config. library** | Config. sequence

```
// External clock configuration
class osync_config_clk_ext extends osync_config;
  constraint clk_ext { clk_sel == osync_config::EXT; };
endclass : osync_config_clk_ext

// Internal clock configuration
class osync_config_clk_int extends osync_config;
  constraint clk_int { clk_sel == osync_config::INT; };
endclass : osync_config_clk_int

// Nominal-case configuration
class osync_config_nominal extends osync_config_clk_int;
  constraint nominal { osync_en == osync_config::ENA;
                       clk_edge == osync_config::FALL;
                       intclk_frequency == 8; };
endclass : osync_config_nominal
```

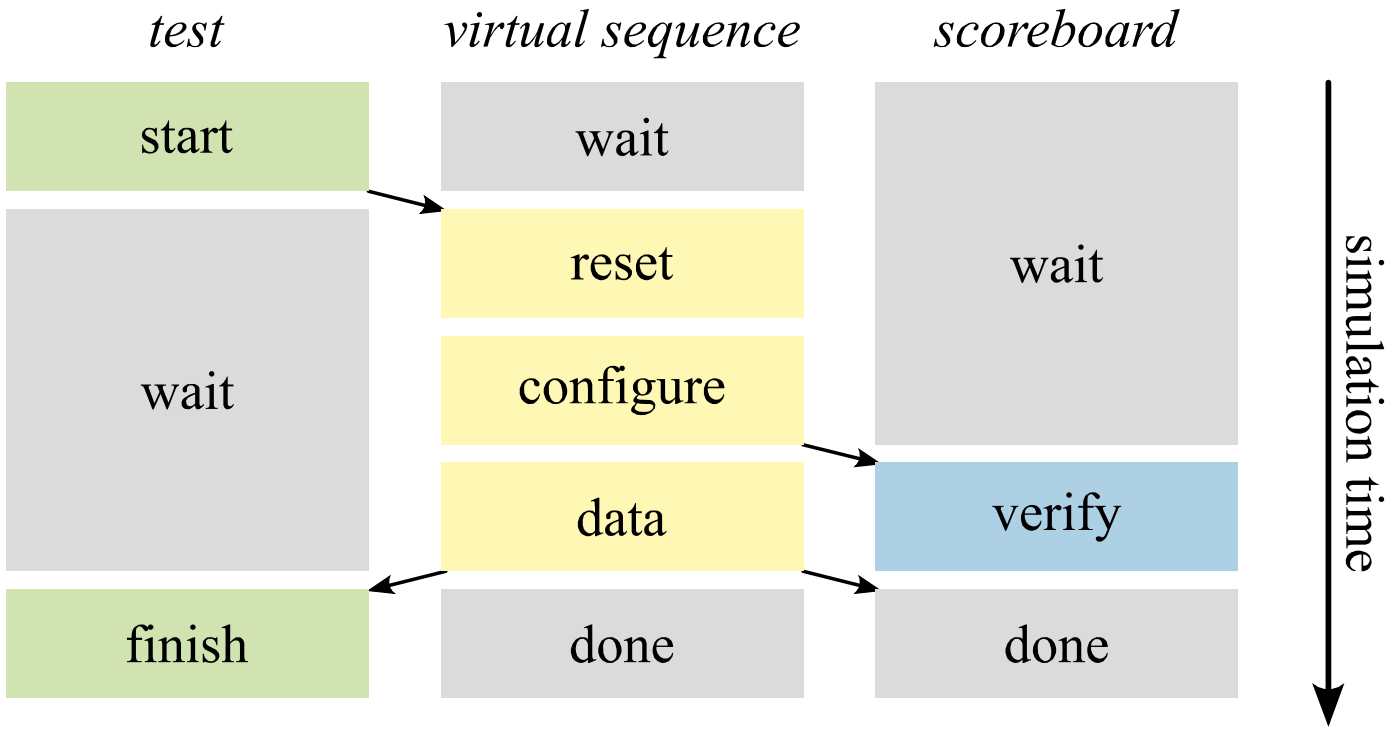
Configuration Mechanism

RAL model | Config. class | Config. library | Config. sequence

```
class osync_ral_cfg_sequence extends uvm_reg_sequence #(apb_sequence_base);
  osync_config cfg;
  ral_sys_osync_regmodel regmodel;
  virtual task pre_start();
    uvm_resource_db #(ral_sys_osync_regmodel)::read_by_name(
                        get_full_name(), "osync_regmodel", regmodel, this);
    cfg = osync_config::type_id::create("cfg");
  endtask
  task body();
    uvm_status_e status;
    // Retrieve block config and write it in the register file
    uvm_resource_db #(osync_config)::read_by_name(
                                    get_full_name(), "osync_cfg", cfg, this);
    regmodel.OSYNC.CFG.write(.status(status),
                .value(cfg.get_cfg()), .path(UVM_FRONTDOOR), .parent(this));
  endtask // body
endclass : osync_ral_cfg_sequence
```

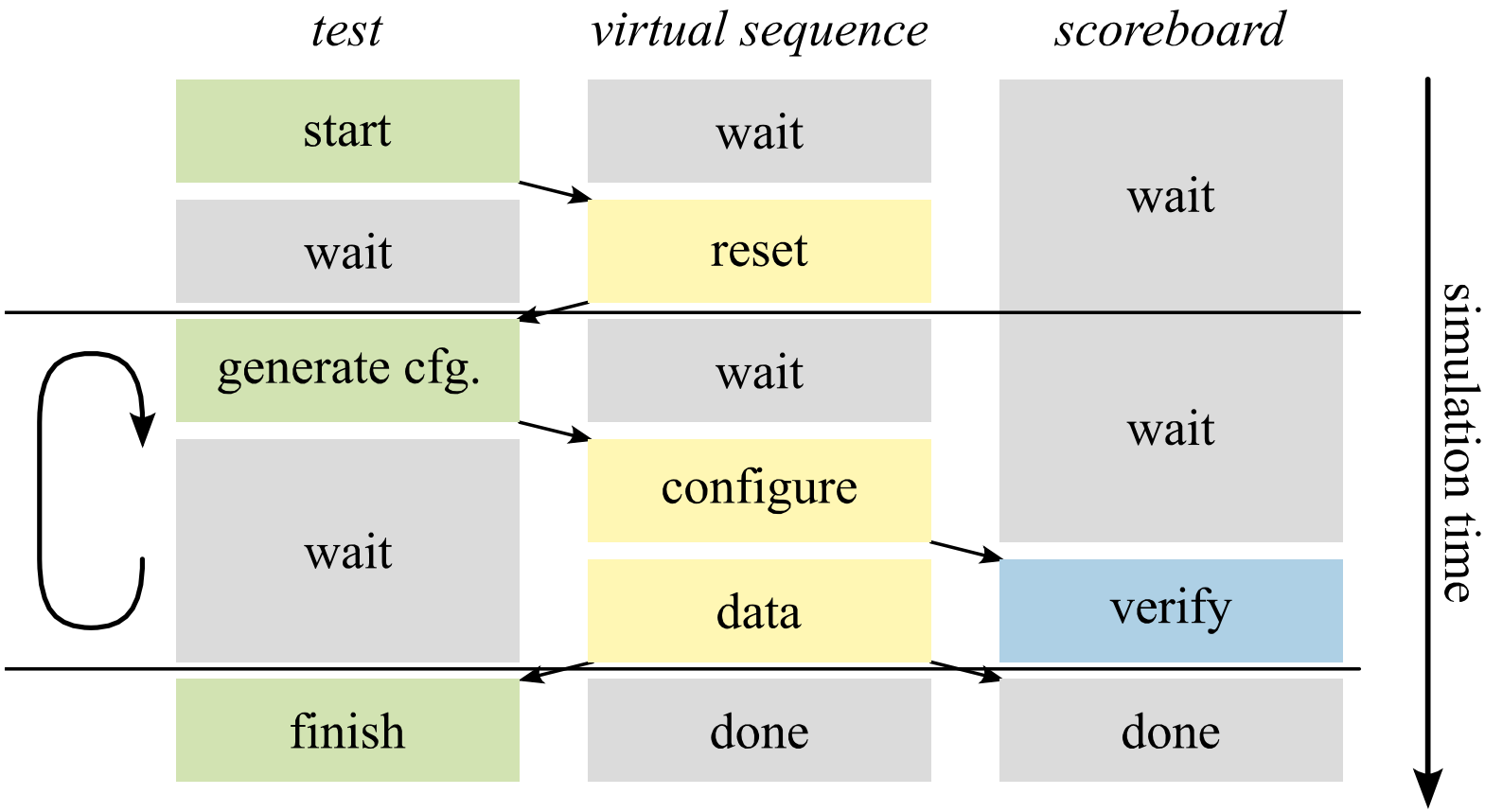
Test Sequences

Basic



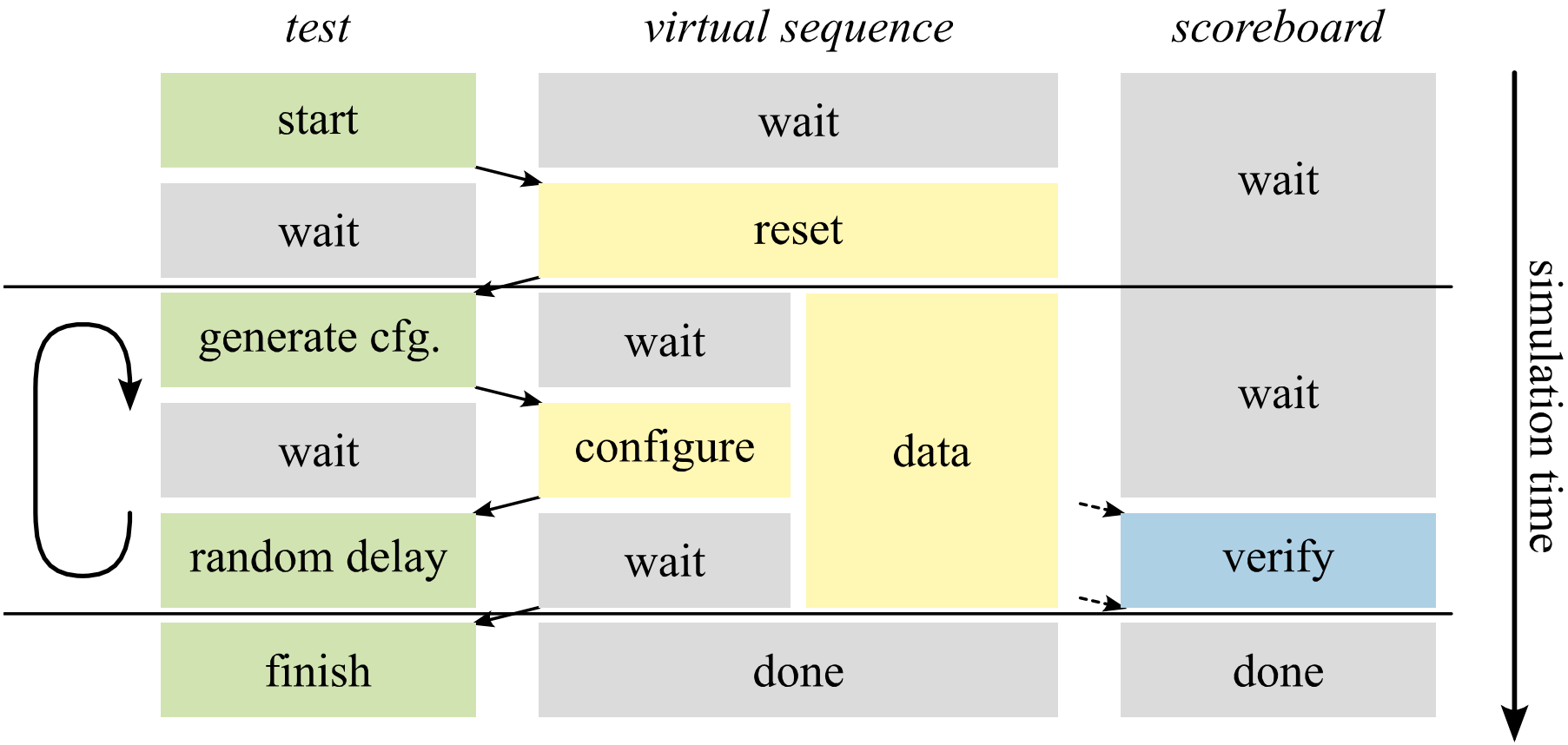
Test Sequences

Static Reconfiguration



Test Sequences

Dynamic Reconfiguration



Interface Verification

SystemVerilog assertion-based protocol checkers
→ Embedded in agents' interfaces

Example:

```
interface ts_interface( input bit clk , input logic rst_n);
  logic         ts_sync;  // TS interface clock signal
  logic [7:0]   ts_data;  // Data byte
  logic         ts_start; // Pulse to signal the beginning of a packet
  logic         ts_valid; // Asserted whenever the data is valid
  logic         ts_stop;  // Asserted by the receiver to stop the sender

  // Check that data values are stable between two sync rising edges
  ts_stable_p: assert property
    (@(posedge clk) disable iff (~rst_n || $past(ts_sync) === 1'bx)
      (~$rose(ts_sync) && ts_valid) |->
             ($stable(ts_valid) && $stable(ts_data) && $stable(ts_start)));

endinterface : ts_interface
```

Coverage

Configurations of hardware blocks

1. Automatic coverage item generation based on RAL
   → No abstraction from configuration register bits

Coverage

Configurations of hardware blocks

1. Automatic coverage item generation based on RAL
   → No abstraction from configuration register bits
2. Coverage items on high-level configuration classes
   → Easy to define, report only interesting information

   Example:

```
covergroup osync_config_cg
             with function sample (osync_config osync_cfg);
  cp_clk_sel   : coverpoint osync_cfg.cfg_clk_sel;
  cp_clk_edge  : coverpoint osync_cfg.cfg_clk_edge;
  cp_intclk    : coverpoint osync_cfg.intclk_frequency {
    bins lo = {[ 1: 5]};  //  1 to  5 MHz
    bins me = {[ 6:10]};  //  6 to 10 MHz
    bins hi = {[11:15]};} // 11 to 15 MHz
  cross_cfg_int : cross cp_clk_sel, cp_clk_edge, cp_intclk;
endgroup
```

Conclusions

- Observations
  - Effort: **~2 weeks** for working environment

    More features added in parallel with test development
  - Several bugs found (one related to reconfiguration)
  - Critical **corner cases** for constant-delay requirement highlighted by **random-constrained** approach

Conclusions

- Observations
  - Effort: **~2 weeks** for working environment

    More features added in parallel with test development
  - Several bugs found (one related to reconfiguration)
  - Critical **corner cases** for constant-delay requirement highlighted by **random-constrained** approach

- Recommendations:
  - **Carefully plan ahead**
  - Exploit object-oriented capabilities
  - Parameterize (even if **not straightforward**)
  - Define “everything” as random

Thank You