VERIFICATION ACADEMY

Advanced OVM (& UVM)

Writing & Managing Tests

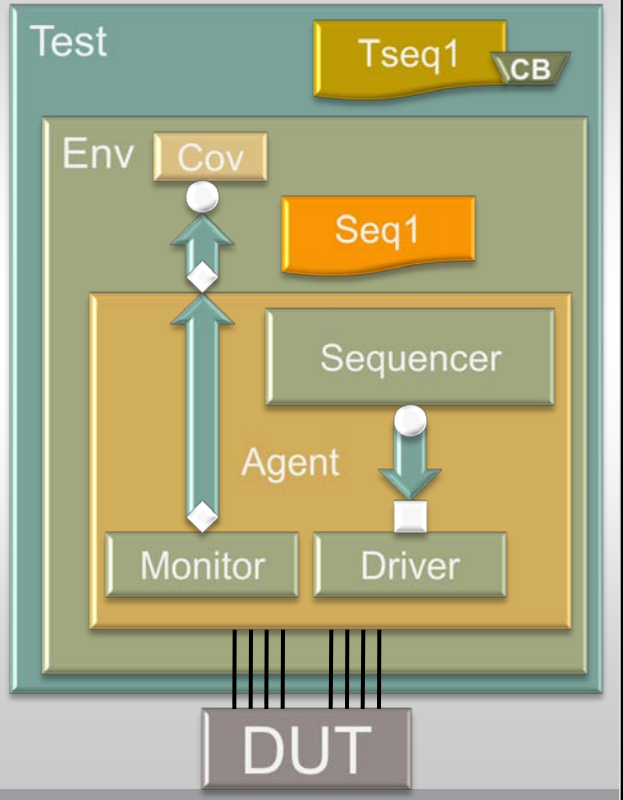
Tom Fitzpatrick
Verification Technologist

academy@mentor.com
www.verificationacademy.com

Mentor Graphics®

What is a Test?

- **The environment is the "Testbench"**
  - Defines what components are needed to verify the DUT
  - Specifies defaults
- **The test's job is to "tweak" the testbench**
  - Configuration
  - Factory overrides
  - Additional sequences
  - Callbacks



Defaults in OVM

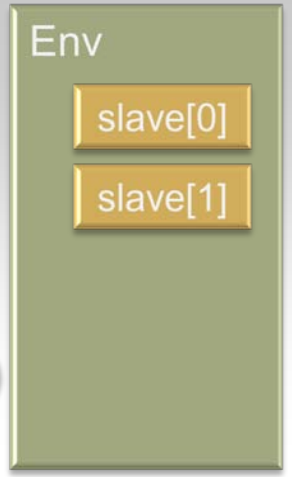
```
class my_env extends ovm_env;
  int nslaves;
  test_seq seq_h
  my_slave slv_h[];
  function void build();
    if(!get_config_int("nslaves",nslaves))
      nslaves = 2;
    slv_h = new[nlsaves];
    for(int i = 0; I < nslaves; i++) begin
      $sformat(name, "slave[%0d]",i);
      slv_h[i] = my_slave::type_id::create(name, this);
    end
  endfunction
  task run();
    seq_h = my_seq::type_id::create("my_seq");
    seq_h.start(slv_h[0]);
  endtask
endclass
```

Always call get_config

If no set_config, use default

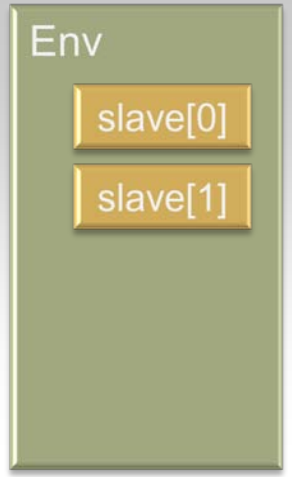
create() sets default type

Start default sequence



Defaults in OVM

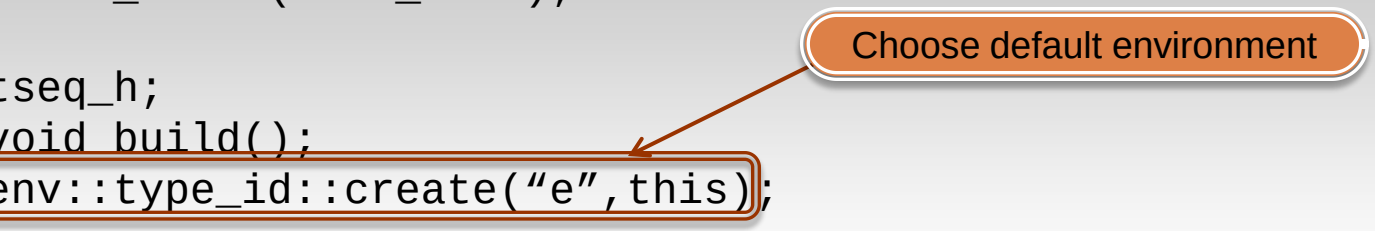
```
class my_env extends ovm_env;
  int nslaves;
  test_seq seq_h;
  my_slave slv_h[];
  function void build();
    if(!get_config_int("nslaves",nslaves))
      nslaves = 2;
    else
      assert({nslaves inside {2,8}})
        else `ovm_error("CFG","Illegal nslaves");
    slv_h = new[nlsaves];

    …
```



Check for legal values!

Use a Base Test to Set Defaults

```
class base_test extends ovm_test;
  `ovm_component_utils(base_test);
  my_env e;
  test_seq tseq_h;
  function void build();
    e = my_env::type_id::create("e",this);
  endfunction

endclass
```



Extend Base Test to Create an Actual Test

```
class my_test1 extends base_test;
  `ovm_component_utils(my_test1);


  function void build();
    super.build();
    set_config_int("e.nslaves",4);
    my_slave::type_id::set_inst_override( err_slave::get_type(),
                                          "e.slave[0]" );|

    test_seq::type_id::set_type_override( test1_seq::get_type());
  endfunction

endclass
```

Get defaults

```
if(!get_config_int("nslaves",nslaves))
      nslaves = 2;
```

Override slave[0] type

Override default sequence type

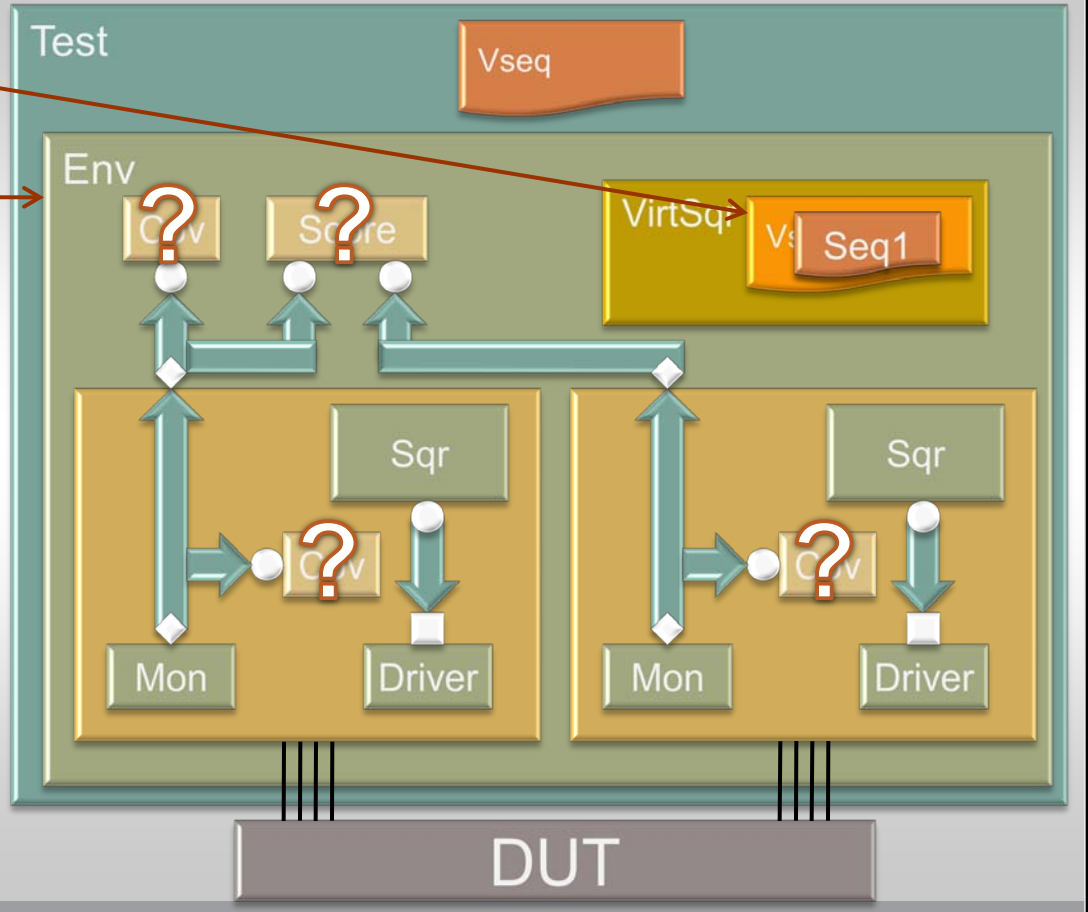
Setup and Invoke Test from Top-Level Module

```
module top;
  ...
  dut_if dut_if1 ();

  initial begin: blk
    dut_if_wrapper if_wrapper = new("if_wrapper", dut_if1 );

    set_config_object("*", "dut_if_wrapper", if_wrapper, 0);


    run_test();
  end

endmodule: top
```

+OVM_TESTNAME=“my_test1”

Complex Environment, Simple Test

- Virtual Sequence embodies “the test”
- Environment sets up agents, etc
- The Test simply chooses the virtual sequence to run
- …and corresponding coverage collectors and scoreboard(s)



Simple Test

```
class seq_test_by_name extends ovm_test;

  `ovm_component_utils(seq_test_by_name);
  my_env e;
  virt_seq vseq_h;
  string seqname;
  function void build();
    e = my_env::type_id::create("e",this);
    if(!$value$plusargs( "SEQNAME=%s" ,seqname))
      vseq_h = virt_seq::type_id::create("vseq");
    else
      $cast(vseq_h, create_object(seqname, "vseq"));
  endfunction
  task run();
     vseq_h.start(e.virt_sequencer);
  endtask
endclass
```

+OVM_TESTNAME="seq_test_by_name"

+SEQNAME="my_virtseq"

Name-based Factory call

Problematic for parameterized classes

Start selected sequence

Extended Test

```
class cov_test extends seq_test_by_name;

  `ovm_component_utils(cov_test);

  function void build();
    super.build();
    agent_cov::type_id::set_inst_override(ctrl_cov::get_type(),
                                          "e.agent1.cov" );
    my_score::type_id::set_type_override( foo_score::get_type());
endfunction

endclass
```

+OVM_TESTNAME="cov_test"

Create Virtual Sequence

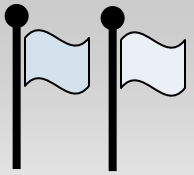
Select Scoreboard to match Virtual Sequence

+SEQNAME="my_fooseq"

Ending the Test

Two Mechanisms for Ending Tests

- **Direct**
  - ovm_test_done objection
  - Components or Sequences can raise or drop objections
  - Test continues until all raised objections are dropped



Two Mechanisms for Ending Tests

- Direct
  - ovm_test_done objection
  - Components or Sequences can raise or drop objections
  - Test continues until all raised objections are dropped
- **Indirect**
  - global_stop_request
  - Test calls global_stop_request when stimulus is complete
  - enable_stop_interrupt
  - Allows components to prevent test from ending until they're ready

Global_stop_request and Objections

- **When all raised objections are dropped, global_stop_request() is called implicitly**
  - All components' stop() tasks are called (in parallel)
- **When global_stop_request() is called explicitly**
  - Wait until all objections are dropped
  - All component's stop() tasks are called

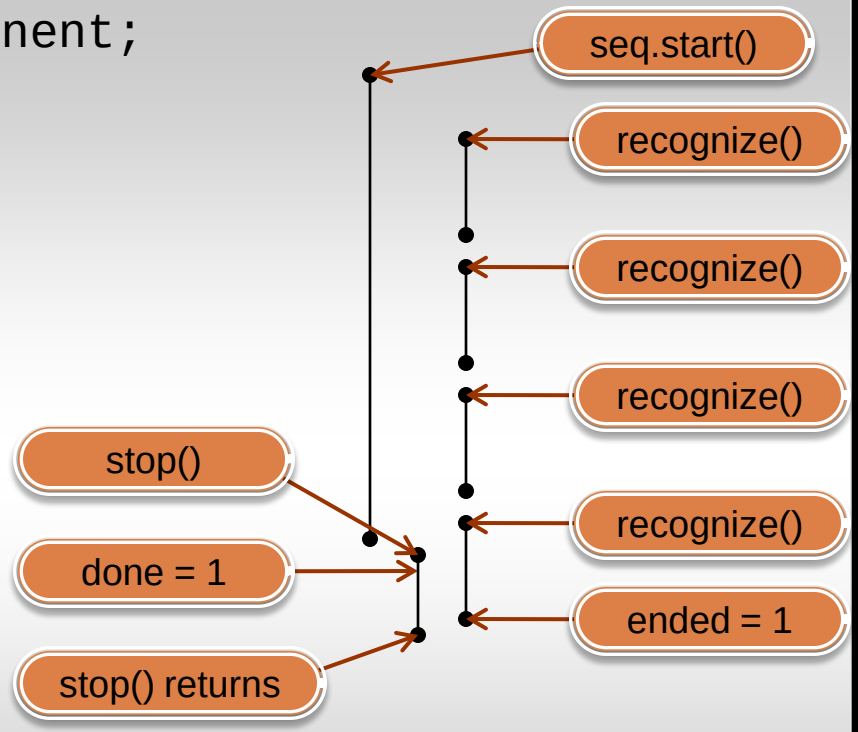
Ending from the Test

```
class my_test extends ovm_test;
  ...
  task run;
    vseq_h.start(e.virt_sequencer);
    global_stop_request();
  endtask
endclass
```

Requires that vseq_h eventually returns

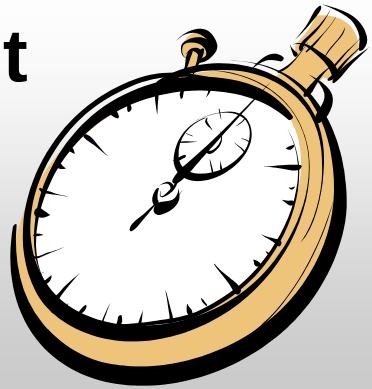
Implementing stop()

```
class my_mon extends ovm_component;
  ...
  task run();
    enable_stop_interrupt = 1;
    while (done = 0) begin
      recognize(tr);
      ap.write(tr);
      end
    ended = 1;
  endtask
  task stop(string ph_name);
    if (ph_name == “run”)
      done = 1;
      wait(ended);
  endtask
endclass
```



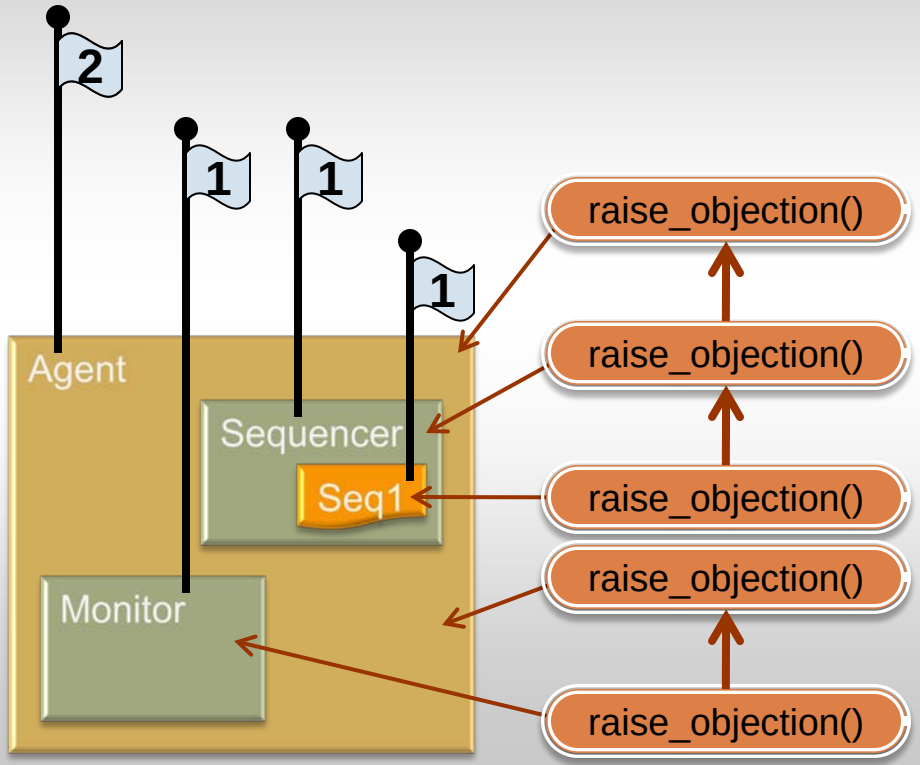
What if stop() Never Returns?

- **OVM includes two watchdog timers**
- **set_global_timeout(time t)**
  - Task-based phase must complete within t
- **set_global_stop_timeout(time t)**
  - Stop() must return within t
- **Setting these to 0 uses the default**
  - Maximum simulation time



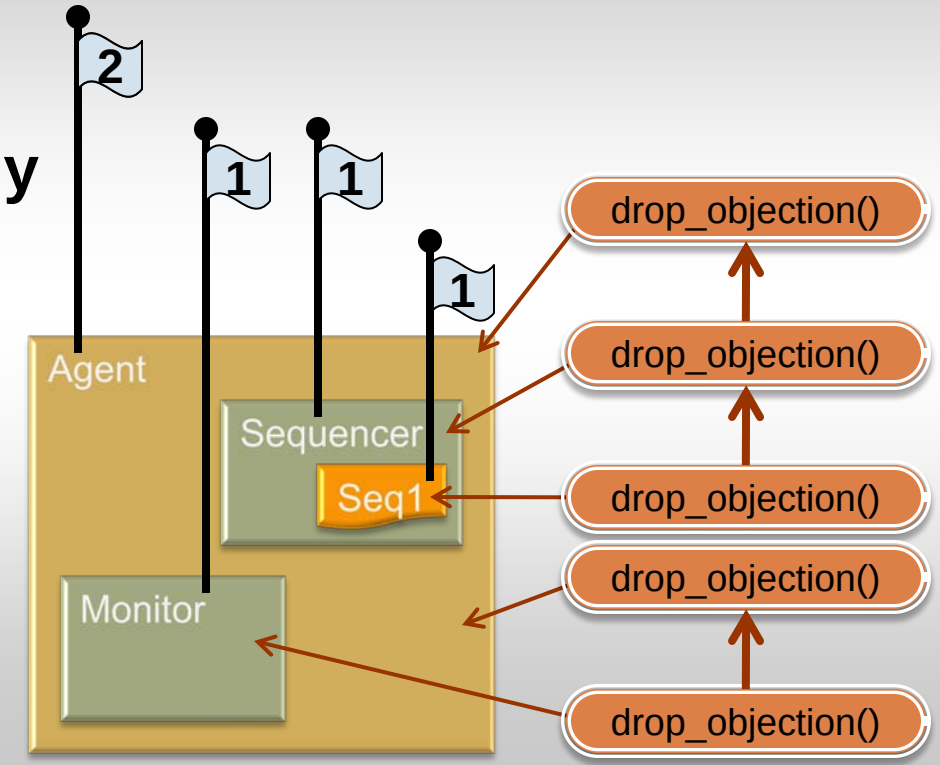
Objections are Hierarchical

- **Objections are raised up the hierarchy**



Objections are Hierarchical

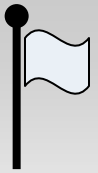
- Objections are raised up the hierarchy
- Objections are dropped hierarchically too
- When a component's count = 0, wait for drain_time to elapse



Using ovm_test_done

```
class my_mon extends ovm_component;
  ...
  task run();
    forever begin
      wait(tx_start);
      ovm_test_done.raise_objection(this);
      collect(tr);
      ap.write(tr);
      ovm_test_done.drop_objection(this);
      end
  endtask
endclass
```

When transaction starts



What if my_mon is the only objector?

Test ends when all objections dropped

Possible Solution

```
class my_env extends ovm_env;
  ...
  function void build();
    ovm_test_done.set_drain_time(ovm_top, 10);
    …
  endfunction
endclass
```

Wait before propagating drop

Recommended

```
class my_test extends ovm_test;
  ...
  task run;
    ovm_test_done.raise_objection(this);
    vseq_h.start(e.virt_sequencer);
    ovm_test_done.drop_objection(this);
  endtask
endclass
```

Don't let a component be the only objector

Requires that vseq_h eventually returns

OVM 2.1.1/UVM Objection API Difference

```
ovm_objection::raise_objection (ovm_object obj=null,                          int count=1);
uvm_objection::raise_objection (uvm_object obj=null, string description="", int count=1);
```

Always use raise_objection(this); for UVM compatability

```
ovm_objection::drop_objection (ovm_object obj=null,                          int count=1);
uvm_objection::drop_objection (uvm_object obj=null, string description="", int count=1);
```

Always use drop_objection(this); for UVM compatability

```
ovm_objection::raised (ovm_object obj, ovm_object src_obj,                  int count);
uvm_objection::raised (uvm_object obj, uvm_object src_obj, string descr, int count);

ovm_objection::dropped (ovm_object obj, ovm_object src_obj,                  int count);
uvm_objection::dropped (uvm_object obj, uvm_object src_obj, string descr, int count);

ovm_objection::all_dropped (ovm_object obj, ovm_object src_obj,                  int count);
uvm_objection::all_dropped (uvm_object obj, uvm_object src_obj, string descr, int count);
```

Callbacks

What are Callbacks?

- Optional OVM/UVM facility to allow test writers to augment component behavior
- Requires Component Developer to design infrastructure
- To be reusable, callbacks must be order-independent

Component Developer Derives Callback Class

```
virtual class bus_driver_cb extends ovm_callback;

  virtual function bit trans_rcv(bus_dr drv, bus_tr tr);
    return 0;
  endfunction

  virtual task trans_exec(bus_dr drv, bus_tr tr);
  endtask

  static string type_name = "bus_driver_cb";

  virtual function string get_type_name();
    return type_name;
  endfunction
endclass
```

Callback method may be a function…

…or a task

Component Developer Puts Callback Hooks in

```
virtual class bus_driver extends ovm_driver #(bus_tr);
  …
  task run();
    forever begin
      seq_item_port.get(tx);
      if(! trans_rcv(tx))
          `ovm_error("CB", "trans_rcv_err");
      < do req bus cycle >
      trans_exec(tx);
    end
  endtask
endclass
```

When transaction is received

After it is executed

Registered Callback Methods Called via Macro

```
virtual class bus_driver extends ovm_driver #(bus_tr);
  …
  virtual function bit trans_rcv(bus_tr tr);
    return `ovm_do_callbacks_exit_on(bus_driver,
                                     bus_driver_cb,
                                     trans_rcv(this,tr),0)
  endfunction

  virtual task trans_executed(bus_tr tr);
    `ovm_do_callbacks(bus_driver,
                      bus_driver_cb,
                      trans_exec(this,tr))
  endtask
  task run();
endclass
```

cycle through function callbacks

Return when first callback returns this value

Type associated with callback

Type of callback object

cycle through callbacks

Method call

User Extends Callback Object

```
class my_bus_driver_cb extends bus_driver_cb;
  …
  virtual task trans_exec(bus_dr drv, bus_tr tr);
    // do something
  endtask

  virtual function bit trans_rcv(bus_dr drv, bus_tr tr);
    trans_rcv = check(tr.addr);
  endfunction

  virtual function string get_type_name();
    return "my_bus_driver_cb";
  endfunction

endclass
```

Define custom functionality

Check and/or Modify contents

Useful for debug

User Extends Callback Object

```
class my_bus_driver_cb2 extends bus_driver_cb;
  …
  virtual task trans_exec(bus_dr drv, bus_tr tr);
    // do something else
  endtask

  virtual function string get_type_name();
    return "my_bus_driver_cb2";
  endfunction

endclass
```

Other custom functionality

User Adds Callback Objects to Components

```
class my_cb_test extends ovm_test;
  my_cb_env e;
  typedef ovm_callbacks #(bus_driver,bus_driver_cb) cbs_t;
  function void build();
    my_bus_driver_cb cb1 = new("cb1");
    my_bus_driver_cb2 cb2 = new("cb2");
    e = my_cb_env::type_id::create("e",this);
  endfunction

  function void end_of_elaboration();
    cbs_t cbs = cbs_t::get_global_cbs();
    cbs.add( e.agent_h.driver_h, cb1 );
    cbs.add( e.agent_h.driver_h, cb2 );
  endfunction
endclass
```

Instantiate callback objects

Components must exist

add callback object…

…to component

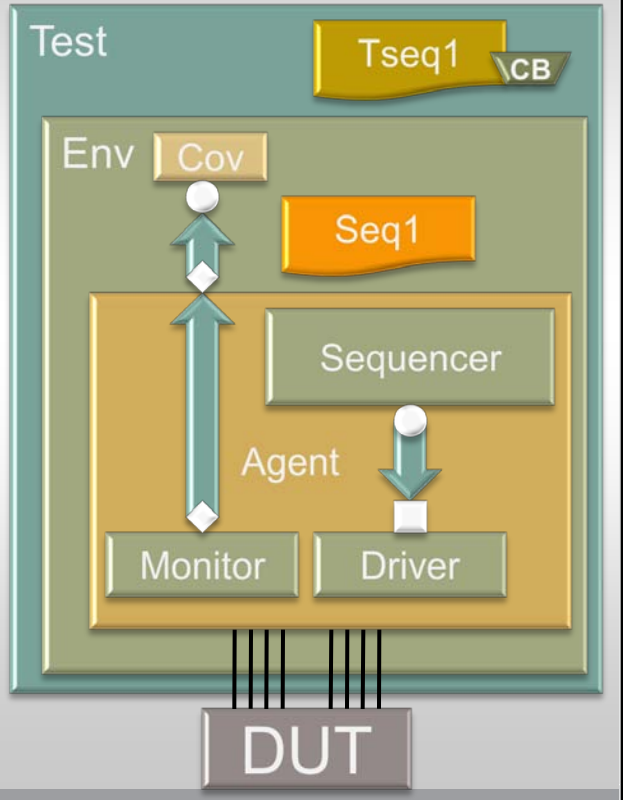
User Adds Callback Objects to Components

```
class my_cb_test extends ovm_test;
  my_cb_env e;
  typedef uvm_callbacks #(bus_driver,bus_driver_cb) cbs_t;
  function void build();
    my_bus_driver_cb cb1 = new("cb1");
    my_bus_driver_cb2 cb2 = new("cb2");
    e = my_cb_env::type_id::create("e",this);
  endfunction

  function void end_of_elaboration();

    cbs_t::add( e.agent_h.driver_h, cb1 );
    cbs_t::add( e.agent_h.driver_h, cb2 );
  endfunction
endclass
```

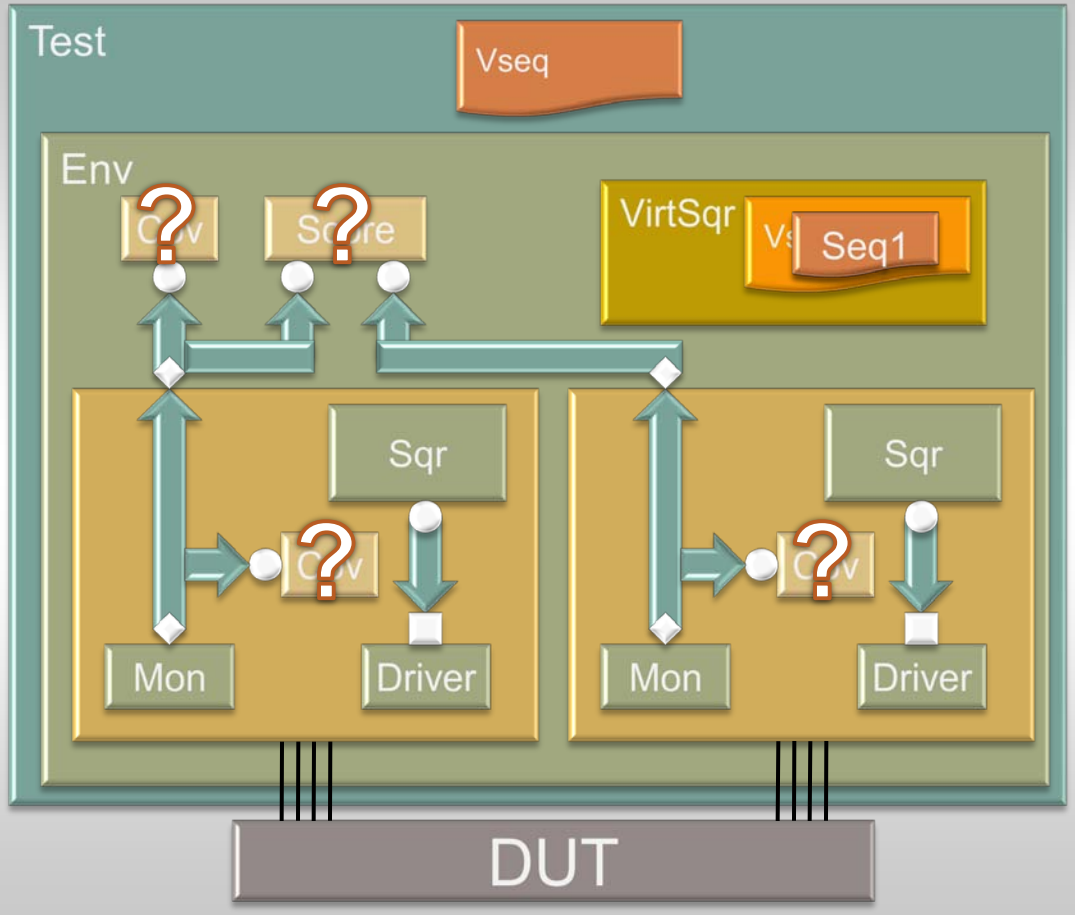
static method

Summary

- **The environment is the “Testbench”**
  - Defines what components are needed to verify the DUT
  - Specifies defaults
- **The test’s job is to “tweak” the testbench**
  - Configuration
  - Factory overrides
  - Additional sequences
  - Callbacks



Summary



VERIFICATION ACADEMY

Advanced OVM (& UVM)

Writing & Managing Tests

Tom Fitzpatrick
Verification Technologist

academy@mentor.com
www.verificationacademy.com

Mentor Graphics®