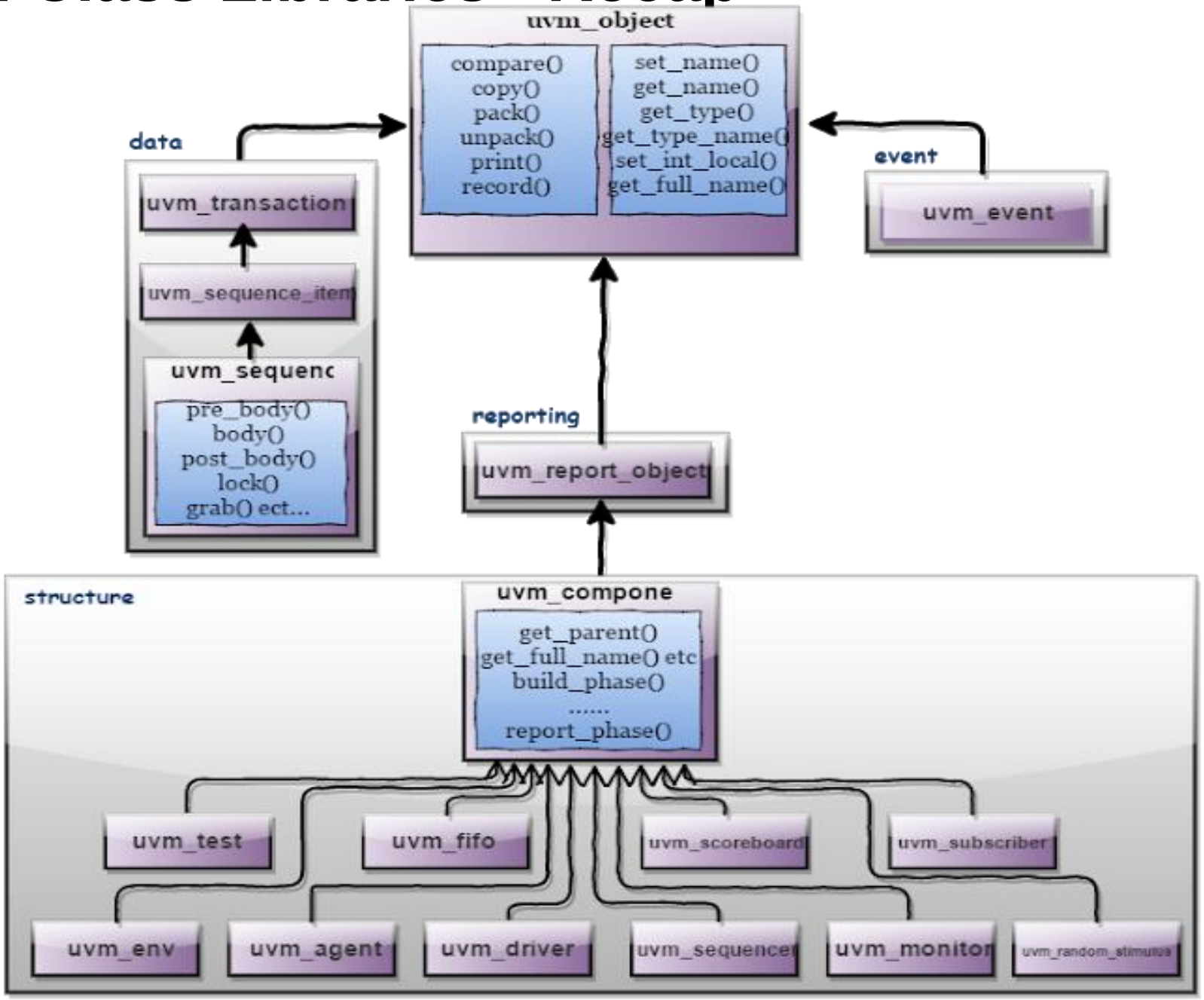
UVM sequencer

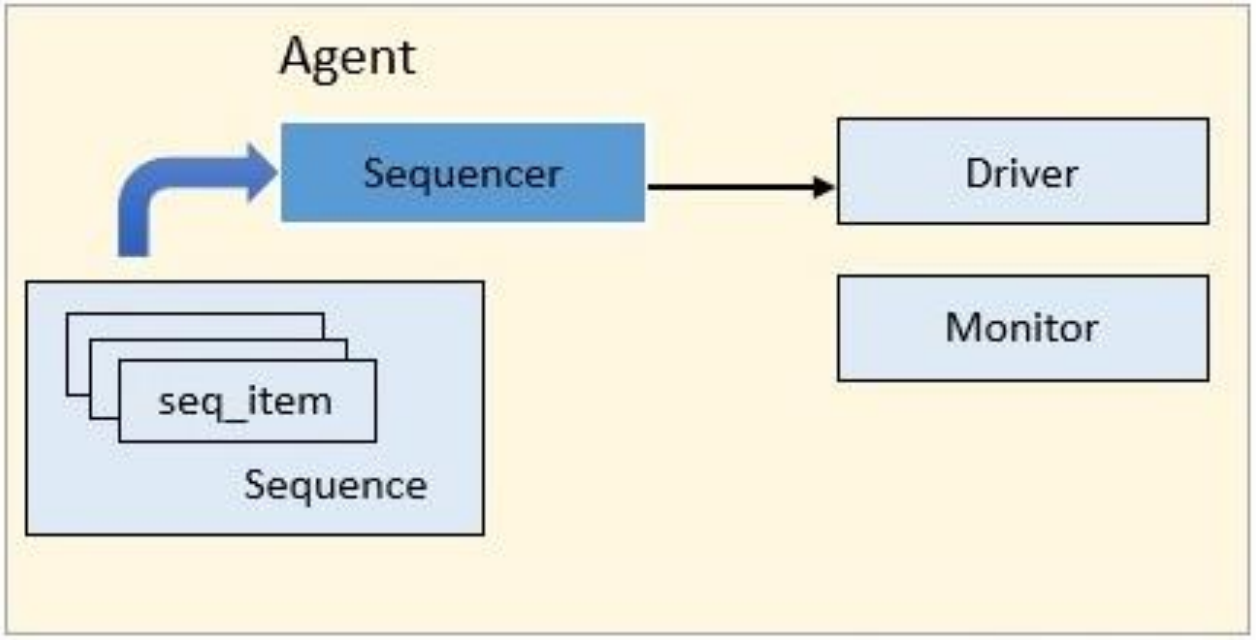
- Reference
- Universal Verification Methodology UVM Cookbook by Siemens Digital Industries Software

UVM Class Libraries - Recap



UVM sequencer

- The sequencer is a mediator who establishes a connection between sequence and driver.
- It passes transactions or sequence items to the driver so that they can be driven to the DUT.



UVM sequencer

- class uvm_sequencer #(type REQ = uvm_sequence_item, RSP = REQ)
  extends uvm_sequencer_param_base #(REQ, RSP)

- A user-defined sequencer is recommended to extend from the parameterized base class “uvm_sequencer” which is parameterized by request (REQ) and response (RSP) item types.
- Response item usage is optional. So, mostly sequencer class is extended from a base class that has only a REQ item.

- class my_sequencer extends uvm_sequencer #(data_item, data_rsp); // with rsp
- class my_sequencer extends uvm_sequencer #(data_item); // without rsp

UVM sequencer

- TLM (Transaction Level Modelling) interface is used by sequencer and driver to pass transactions.
- seq_item_export and seq_item_port TLM connect methods are defined in uvm_sequencer and uvm_driver class.

```
class my_sequencer extends uvm_sequencer #(mem_seq_item);
  `uvm_component_utils(my_sequencer)
   function new (string name, uvm_component parent);
     super.new(name, parent);
   endfunction
endclass
```

m_sequencer and p_sequencer

- m_sequencer
- m_sequencer is a handle available by default in a sequence. m_sequencer has a type of uvm_sequencer_base.
- Simply, it is a reference handle to the sequencer on which the sequence is running.
- To run a sequence on the sequencer, the start() method is called. This start method needs to provide a sequencer handle.
- Ex:
- base_seq.start(env_o.seqr);
- Here, m_seqeuncer is a handle for base_seq that is set to env_o.seqr.

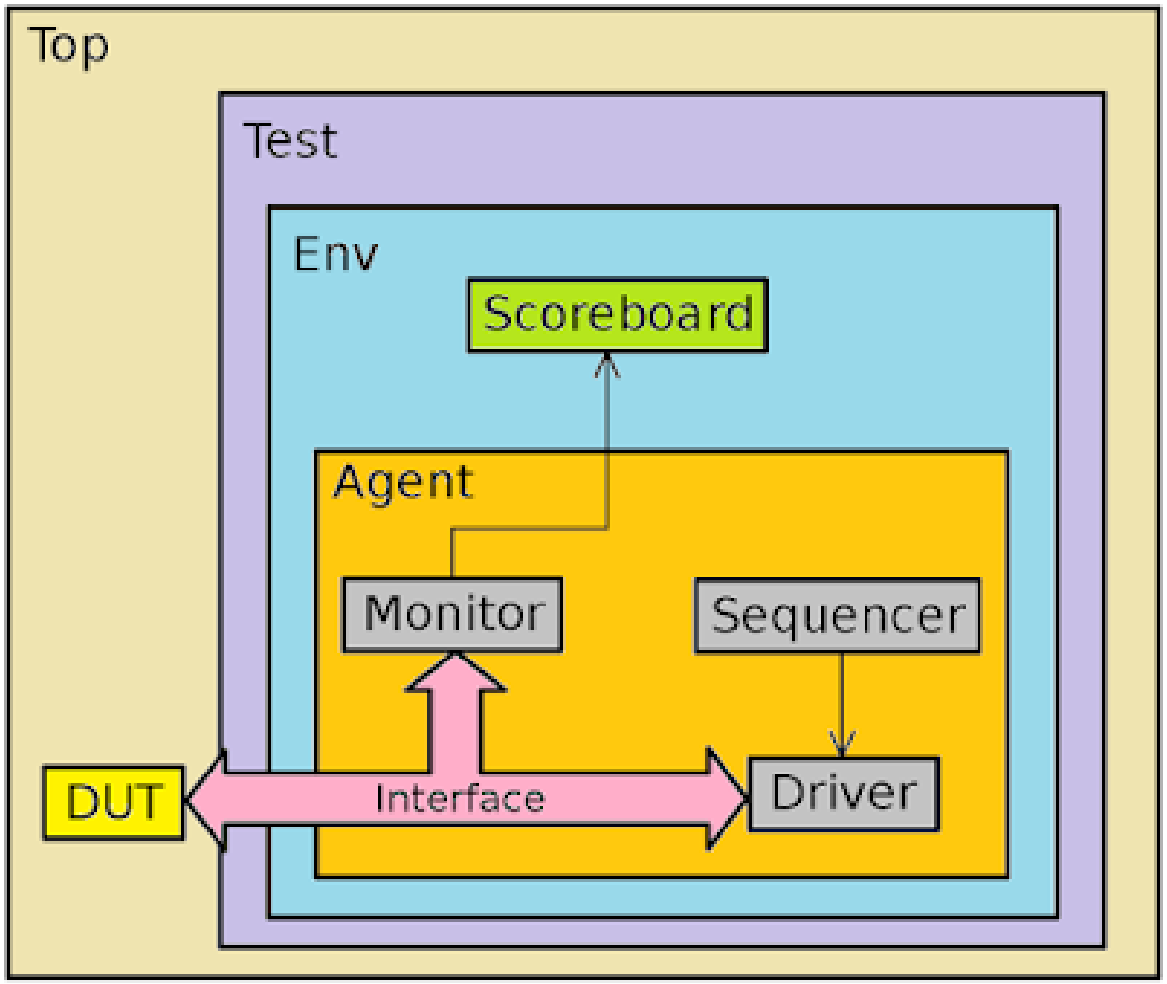
m_sequencer and p_sequencer

- p_sequencer
- All sequences have a m_sequencer handle but they do not have a p_sequencer handle.
- p_sequencer is not defined automatically. It is defined using macro `uvm_declare_p_sequencer(sequencer_name).

```
`define uvm_declare_p_sequencer(SEQUENCER)
   SEQUENCER p_sequencer;
   virtual function void m_set_p_sequencer();
     super.m_set_p_sequencer();
     if( !$cast(p_sequencer, m_sequencer))
       `uvm_fatal("DCLPSQ", $sformatf("%m %s Error casting
        p_sequencer, please verify that this sequence/sequence item
        is intended to execute on this type of
        sequencer",get_full_name()))
   endfunction
```

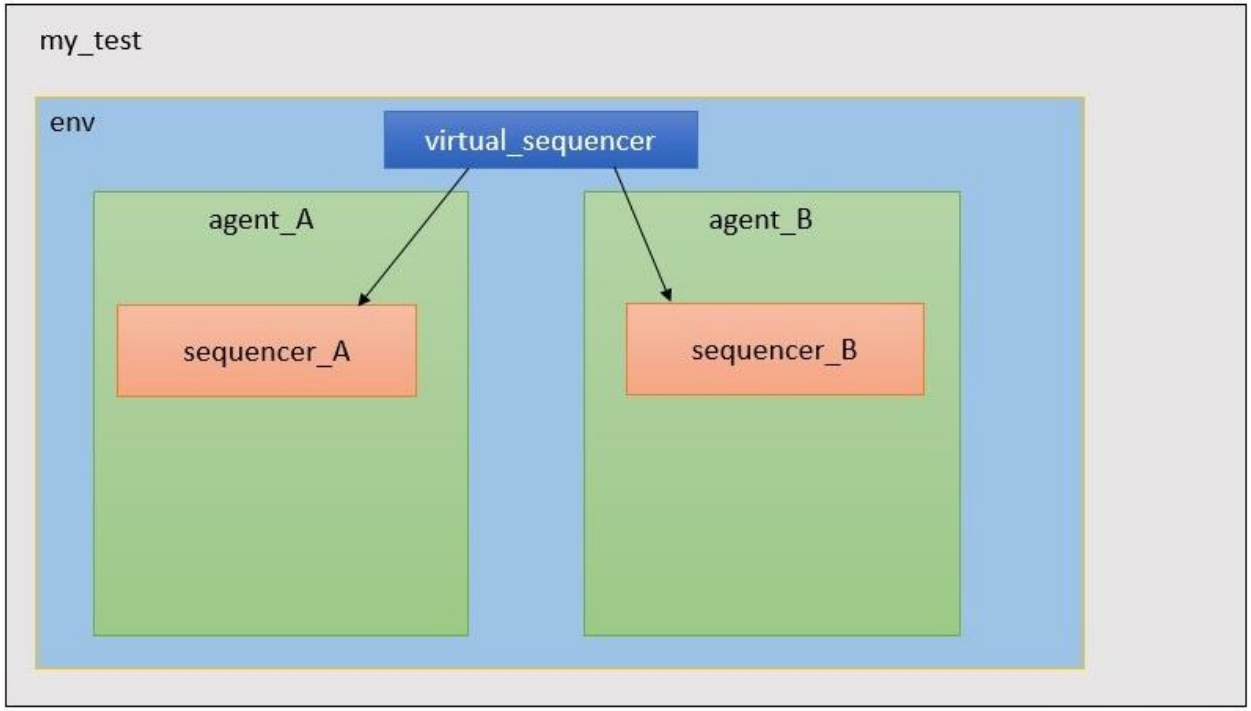
- Define p_sequencer macro does declare p_sequencer handle of SEQUENCER type and casts m_sequencer handle to p_sequencer

UVM Testbench Architecture



Virtual Sequence and Virtual Sequencers

- A virtual sequence is a container that starts multiple sequences on different sequencers



- Different agents to handle the different protocols
- Therefore, there is a need to execute sequences on corresponding sequencers

Virtual Sequence and Virtual Sequencers

- A virtual sequence is usually executed on the virtual sequencer.
- A virtual sequence gives control to start different sequences.
- If there are multiple agents and stimulus coordination required, use a virtual sequencer.
  - virtual_sequence and virtual_sequencer do not require any virtual keyword.
- UVM does not have uvm_virtual_sequence and uvm_virtual_sequencer as base classes.
- A virtual sequence is derived from uvm_sequence.
- A virtual_sequencer is derived from uvm_sequencer as a base class.

Example- Without virtual sequence and virtual sequencer

```
// No Virtual Sequencer
class core_A_sequencer extends uvm_sequencer #(seq_item);
   `uvm_component_utils(core_A_sequencer)
    function new(string name = "core_A_sequencer", uvm_component
       parent = null);
       super.new(name, parent);
    endfunction
endclass

class core_B_sequencer extends uvm_sequencer #(seq_item);
  `uvm_component_utils(core_B_sequencer)
   function new(string name = "core_B_sequencer", uvm_component
        parent = null);
     super.new(name, parent);
   endfunction
endclass
```

Example- Without virtual sequence and virtual sequencer

```
// base_test
class base_test extends uvm_test;
env env_o; core_A_seq Aseq; core_B_seq Bseq;
`uvm_component_utils(base_test)
   function new(string name = "base_test",uvm_component parent=null);
     super.new(name, parent);
   endfunction
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env_o = env::type_id::create("env_o", this);
  endfunction
  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    Aseq = core_A_seq::type_id::create("Aseq");
    Bseq = core_B_seq::type_id::create("Bseq");
    Aseq.start(env_o.agt_A.seqr_A);
    Bseq.start(env_o.agt_B.seqr_B);
    phase.drop_objection(this);
  endtask
endclass
```

Example- Without virtual sequence and virtual sequencer

```
UVM_INFO sequence.sv(10) @ 0: uvm_test_top.env_o.agt_A.seqr_A@@Aseq
[core_A_seq] core_A_seq: Inside Body

UVM_INFO driver.sv(38) @ 0: uvm_test_top.env_o.agt_A.drv_A
[core_A_driver] Driving from core A

UVM_INFO sequence.sv(30) @ 50: uvm_test_top.env_o.agt_B.seqr_B@@Bseq
[core_B_seq] core_B_seq: Inside Body

UVM_INFO driver.sv(55) @ 50: uvm_test_top.env_o.agt_B.drv_B
[core_B_driver] Driving from core B
```

Example- With virtual sequence and without virtual sequencer

```
// virtual sequence
class virtual_seq extends uvm_sequence #(seq_item);
  core_A_seq Aseq; core_B_seq Bseq; core_A_sequencer seqr_A;
  core_B_sequencer seqr_B;
  `uvm_object_utils(virtual_seq)
   function new (string name = "virtual_seq");
     super.new(name);
   endfunction

   task body();
     `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
      Aseq = core_A_seq::type_id::create("Aseq");
      Bseq = core_B_seq::type_id::create("Bseq");
      Aseq.start(seqr_A);
      Bseq.start(seqr_B);
   endtask
endclass
```

Example- With virtual sequence and without virtual sequencer

```
// No Virtual Sequencer
class core_A_sequencer extends uvm_sequencer #(seq_item);
   `uvm_component_utils(core_A_sequencer)
    function new(string name = "core_A_sequencer", uvm_component
       parent = null);
       super.new(name, parent);
    endfunction
endclass

class core_B_sequencer extends uvm_sequencer #(seq_item);
  `uvm_component_utils(core_B_sequencer)
   function new(string name = "core_B_sequencer", uvm_component
        parent = null);
     super.new(name, parent);
   endfunction
endclass
```

Example- With virtual sequence and without virtual sequencer

```
UVM_INFO sequence.sv(56) @ 0: reporter@@v_seq [virtual_seq]
virtual_seq: Inside Body

UVM_INFO sequence.sv(10) @ 0: uvm_test_top.env_o.agt_A.seqr_A@@Aseq
[core_A_seq] core_A_seq: Inside Body

UVM_INFO driver.sv(38) @ 0: uvm_test_top.env_o.agt_A.drv_A
[core_A_driver] Driving from core A

UVM_INFO sequence.sv(30) @ 50: uvm_test_top.env_o.agt_B.seqr_B@@Bseq
[core_B_seq] core_B_seq: Inside Body

UVM_INFO driver.sv(55) @ 50: uvm_test_top.env_o.agt_B.drv_B
[core_B_driver] Driving from core B
```

Example- With virtual sequence and virtual sequencer using p_senquencer handle

```
// Virtual sequence
class virtual_seq extends uvm_sequence #(seq_item);
core_A_seq Aseq; core_B_seq Bseq;
core_A_sequencer seqr_A; core_B_sequencer seqr_B;
`uvm_object_utils(virtual_seq)
`uvm_declare_p_sequencer(virtual_sequencer)
  function new (string name = "virtual_seq");
    super.new(name);
  endfunction
  task body();
    `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
     Aseq = core_A_seq::type_id::create("Aseq");
     Bseq = core_B_seq::type_id::create("Bseq");
     Aseq.start(p_sequencer.seqr_A);
     Bseq.start(p_sequencer.seqr_B);
  endtask
endclass
```

Example- With virtual sequence and virtual sequencer using p_senquencer handle

```
// Virtual p_sequencer
class virtual_sequencer extends uvm_sequencer;
`uvm_component_utils(virtual_sequencer)
core_A_sequencer seqr_A;
core_B_sequencer seqr_B;
  function new(string name = "virtual_sequencer", uvm_component
                  parent = null);
     super.new(name, parent);
  endfunction
endclass
```

Example- With virtual sequence and virtual sequencer using p_senquencer handle

```
UVM_INFO sequence.sv(56) @ 0: uvm_test_top.env_o.v_seqr@@v_seq
[virtual_seq] virtual_seq: Inside Body

UVM_INFO sequence.sv(10) @ 0: uvm_test_top.env_o.agt_A.seqr_A@@Aseq
[core_A_seq] core_A_seq: Inside Body

UVM_INFO driver.sv(38) @ 0: uvm_test_top.env_o.agt_A.drv_A
[core_A_driver] Driving from core A

UVM_INFO sequence.sv(30) @ 50: uvm_test_top.env_o.agt_B.seqr_B@@Bseq
[core_B_seq] core_B_seq: Inside Body

UVM_INFO driver.sv(55) @ 50: uvm_test_top.env_o.agt_B.drv_B
[core_B_driver] Driving from core B
```

Example- With virtual sequence and virtual sequencer but without using p_senquencer handle

```
// virtual sequence
class virtual_seq extends uvm_sequence #(seq_item);
core_A_seq Aseq; core_B_seq Bseq;
core_A_sequencer seqr_A; core_B_sequencer seqr_B;
`uvm_object_utils(virtual_seq)
  function new (string name = "virtual_seq");
    super.new(name);
   endfunction
   task body();
     env env_s;
     `uvm_info(get_type_name(), "virtual_seq: Inside Body", UVM_LOW);
      Aseq = core_A_seq::type_id::create("Aseq");
      Bseq = core_B_seq::type_id::create("Bseq");
      // virtual_sequencer is created in env, so we need env handle
         to find v_seqr.
      if(!$cast(env_s, uvm_top.find("uvm_test_top.env_o")))
        `uvm_error(get_name(), "env_o is not found");
         Aseq.start(env_s.v_seqr.seqr_A);
         Bseq.start(env_s.v_seqr.seqr_B);
   endtask
endclass
```

Example- With virtual sequence and virtual sequencer but without using p_senquencer handle

```
// virtual_sequencer
class virtual_sequencer extends uvm_sequencer;
`uvm_component_utils(virtual_sequencer)
core_A_sequencer seqr_A; core_B_sequencer seqr_B;
  function new(string name = "virtual_sequencer", uvm_component  z
                parent = null);
    super.new(name, parent);
  endfunction
endclass
```

Example- With virtual sequence and virtual sequencer but without using p_senquencer handle

```
UVM_INFO sequence.sv(55) @ 0: uvm_test_top.env_o.v_seqr@@v_seq
[virtual_seq] virtual_seq: Inside Body

UVM_INFO sequence.sv(10) @ 0: uvm_test_top.env_o.agt_A.seqr_A@@Aseq
[core_A_seq] core_A_seq: Inside Body

UVM_INFO driver.sv(38) @ 0: uvm_test_top.env_o.agt_A.drv_A
[core_A_driver] Driving from core A

UVM_INFO sequence.sv(30) @ 50: uvm_test_top.env_o.agt_B.seqr_B@@Bseq
[core_B_seq] core_B_seq: Inside Body

UVM_INFO driver.sv(55) @ 50: uvm_test_top.env_o.agt_B.drv_B
[core_B_driver] Driving from core B
```