VERIFICATION ACADEMY

UVM Basics

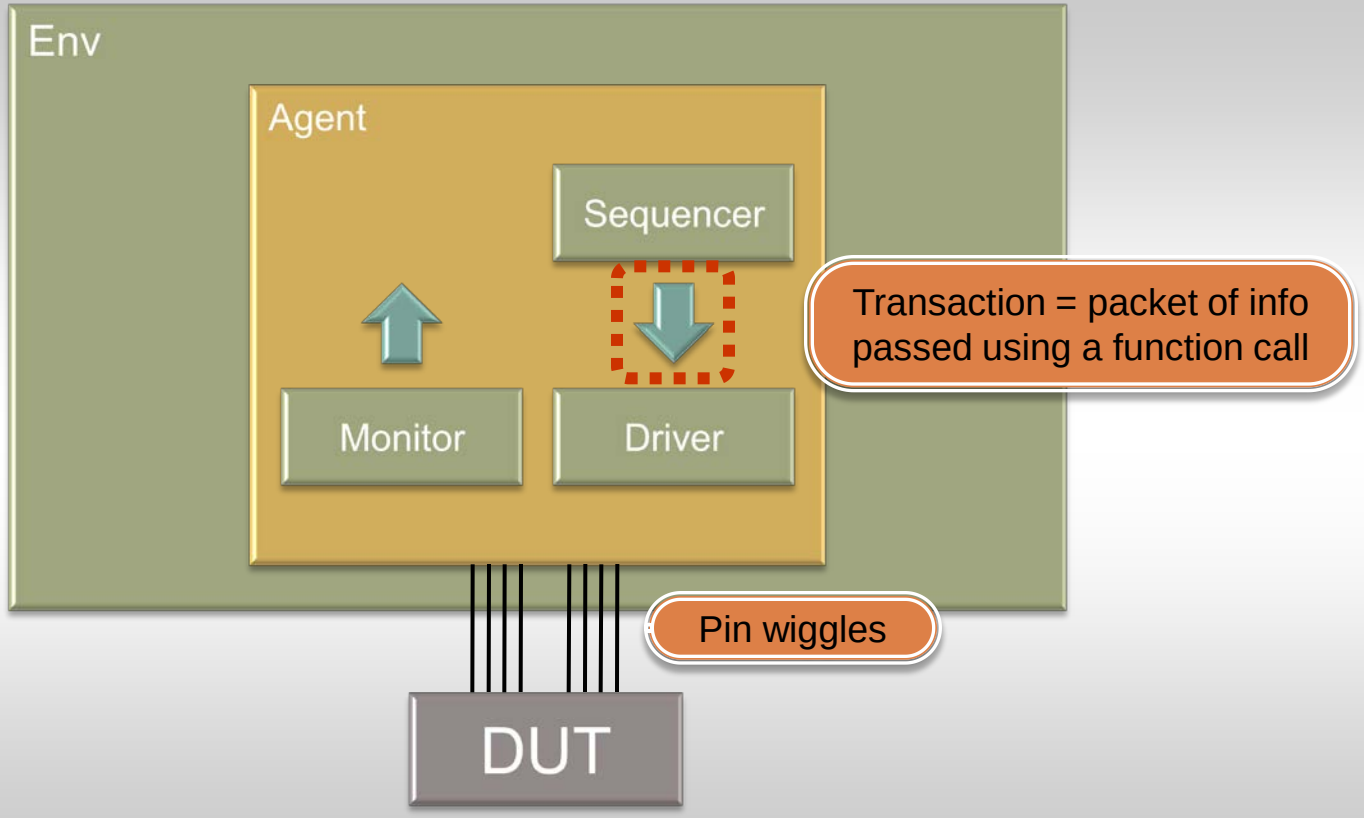
Introducing Transactions

John Aynsley
CTO, Doulos

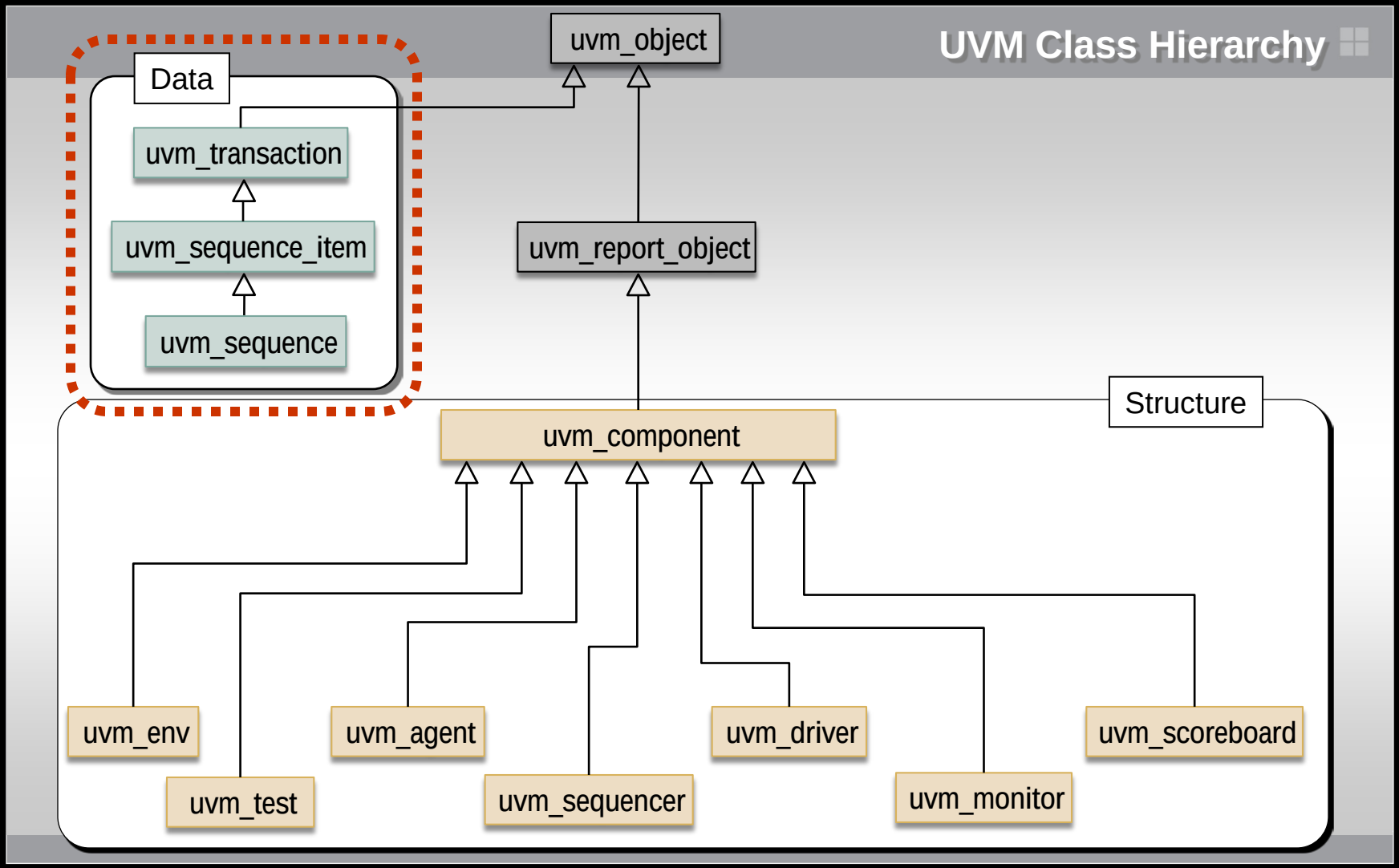
academy@mentor.com
www.verificationacademy.com

Mentor Graphics®

Sequencer and Driver



UVM Class Hierarchy



Transaction

```
class my_transaction extends uvm_sequence_item;
```

Not uvm_transaction

Transaction

```
class my_transaction extends uvm_sequence_item;

   `uvm_object_utils(my_transaction)
```

Not uvm_component_utils

Transaction

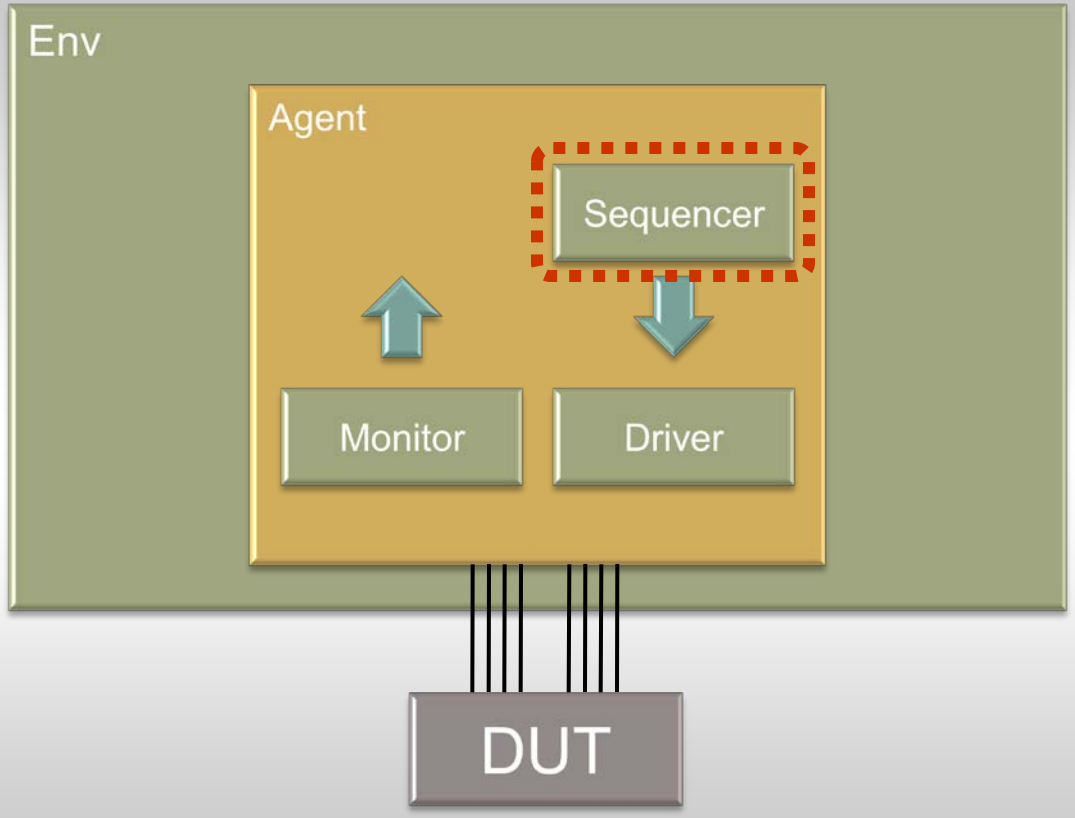
```
class my_transaction extends uvm_sequence_item;

   `uvm_object_utils(my_transaction)

   rand bit cmd;
   rand int addr;
   rand int data;

   constraint c_addr { addr >= 0; addr < 256; }
   constraint c_data { data >= 0; data < 256; }
```

Transaction

```
class my_transaction extends uvm_sequence_item;

   `uvm_object_utils(my_transaction)

   rand bit cmd;
   rand int addr;
   rand int data;

   constraint c_addr { addr >= 0; addr < 256; }
   constraint c_data { data >= 0; data < 256; }

   function new (string name = "");
     super.new(name);
   endfunction: new

 endclass: my_transaction
```

No parent

Sequencer and Driver



uvm_sequencer

```
class my_sequencer extends uvm_sequencer ...
```

uvm_sequencer

```
class my_sequencer extends uvm_sequencer #(my_transaction);
```

uvm_sequencer

```
class my_sequencer extends uvm_sequencer #(my_transaction);

  `uvm_component_utils(my_sequencer)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction: new

endclass: my_sequencer
```

A sequencer is a component

Vanilla Sequencer – use typedef

```
/*
class my_sequencer extends uvm_sequencer #(my_transaction);

  `uvm_component_utils(my_sequencer)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction: new

endclass: my_sequencer
*/

typedef uvm_sequencer #(my_transaction) my_sequencer;
```

uvm_sequence

```
class my_sequence extends uvm_sequence #(my_transaction);
```

uvm_sequence

```
class my_sequence extends uvm_sequence #(my_transaction);

  `uvm_object_utils(my_sequence)
```

uvm_sequence

```
class my_sequence extends uvm_sequence #(my_transaction);

  `uvm_object_utils(my_sequence)

  function new (string name = "");
    super.new(name);
  endfunction: new
```

uvm_sequence

```
class my_sequence extends uvm_sequence #(my_transaction);

  `uvm_object_utils(my_sequence)

  function new (string name = "");
    super.new(name);
  endfunction: new

  task body;
    ...
```

The behavior of the sequence

uvm_sequence

```
task body;
  forever
  begin




  end
endtask: body
```

Loop or no loop?

uvm_sequence

```
task body;
  forever
  begin
    my_transaction tx;
    tx = my_transaction::type_id::create("tx");


  end
endtask: body
```

Step 1

"Factory method" can be overridden in tests

uvm_sequence

```
task body;
  forever
  begin
    my_transaction tx;
    tx = my_transaction::type_id::create("tx");
    start_item(tx);


  end
endtask: body
```

Step 2

Waits for driver

uvm_sequence

```
task body;
  forever
  begin
    my_transaction tx;
    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() );

  end
endtask: body
```

Step 3

Only randomized when driver is ready

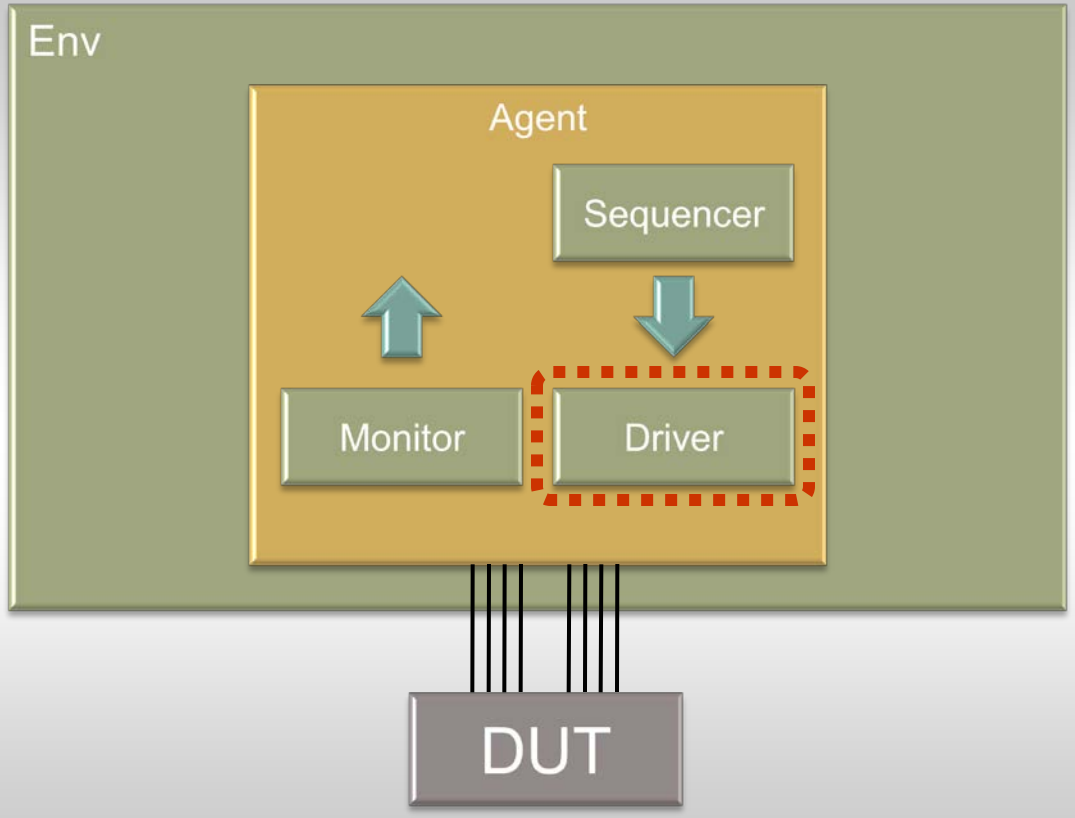
uvm_sequence

```
task body;
  forever
  begin
    my_transaction tx;
    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() );
    finish_item(tx);
  end
endtask: body
```

Step 4

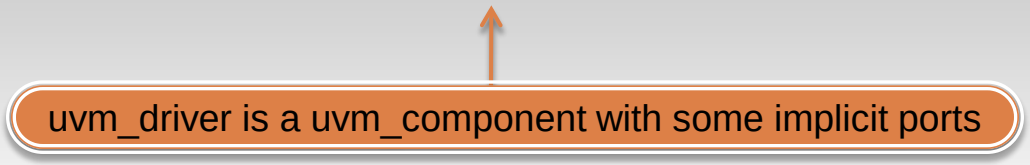
Deliver transaction to driver

Sequencer and Driver



Driver

```
class my_driver extends uvm_driver #(my_transaction);
```



Driver

```
class my_driver extends uvm_driver #(my_transaction);
  `uvm_component_utils(my_driver)

  virtual dut_if dut_vi;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction: new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    ...

  task run_phase(uvm_phase phase);
    ...
```

Driver will access the DUT interface

Driver

```
task run_phase(uvm_phase phase);


  repeat(4)
  begin
    my_transaction tx;
    @(posedge dut_vi.clock);




  end


endtask: run_phase
```

Timing taken from DUT clock

Driver

```
task run_phase(uvm_phase phase);


  repeat(4)
  begin
    my_transaction tx;
    @(posedge dut_vi.clock);
    seq_item_port.get(tx);


  end


endtask: run_phase
```

get() waits on start_item/finish_item

Implicit port connected to sequencer

Driver

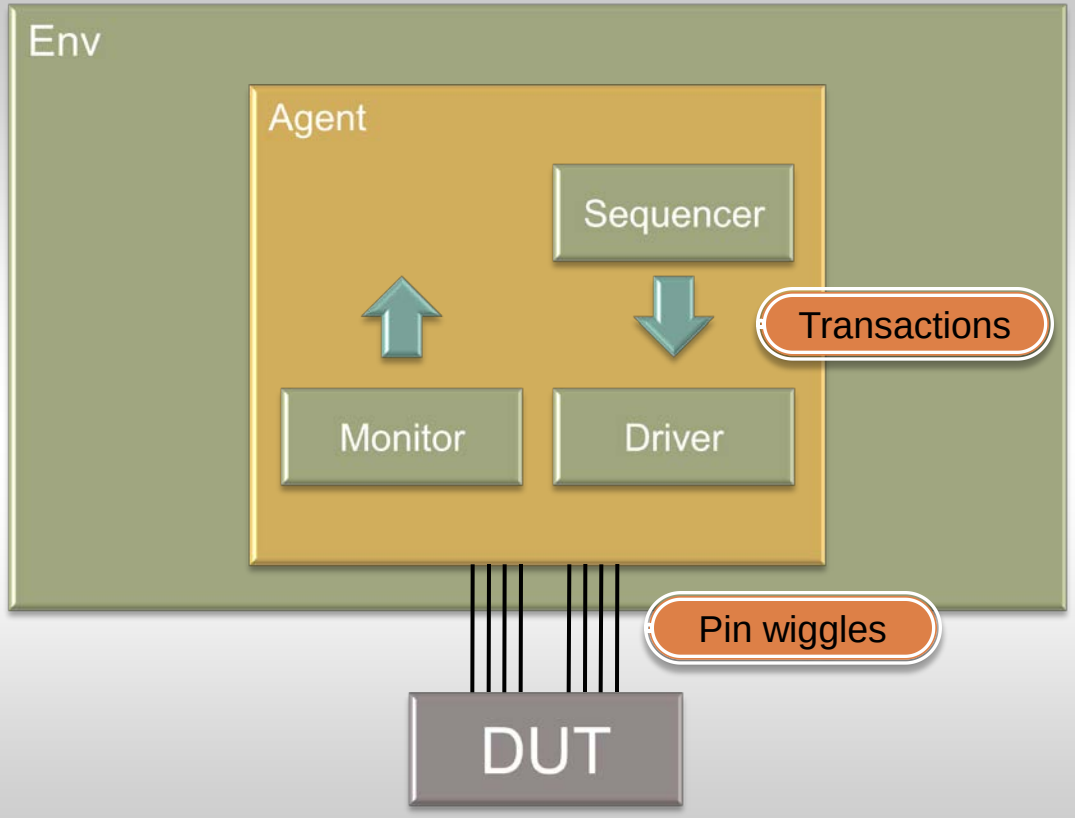
```
task run_phase(uvm_phase phase);


  repeat(4)
  begin
    my_transaction tx;
    @(posedge dut_vi.clock);
    seq_item_port.get(tx);

    dut_vi.cmd  = tx.cmd;
    dut_vi.addr = tx.addr;
    dut_vi.data = tx.data;
  end


endtask: run_phase
```

Pin wiggles

Driver

```
task run_phase(uvm_phase phase);
  phase.raise_objection(this);

  repeat(4)
  begin
    my_transaction tx;
    @(posedge dut_vi.clock);
    seq_item_port.get(tx);

    dut_vi.cmd  = tx.cmd;
    dut_vi.addr = tx.addr;
    dut_vi.data = tx.data;
  end

  @(posedge dut_vi.clock) phase.drop_objection(this);
endtask: run_phase
```

Summary


Env
Agent
Sequencer
Transactions
Monitor
Driver
Pin wiggles
DUT

VERIFICATION ACADEMY

UVM Basics

Introducing Transactions

John Aynsley
CTO, Doulos

academy@mentor.com
www.verificationacademy.com

Mentor Graphics®