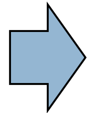
The Finer Points of UVM: Tasting Tips for the Connoisseur

John Aynsley, Doulos

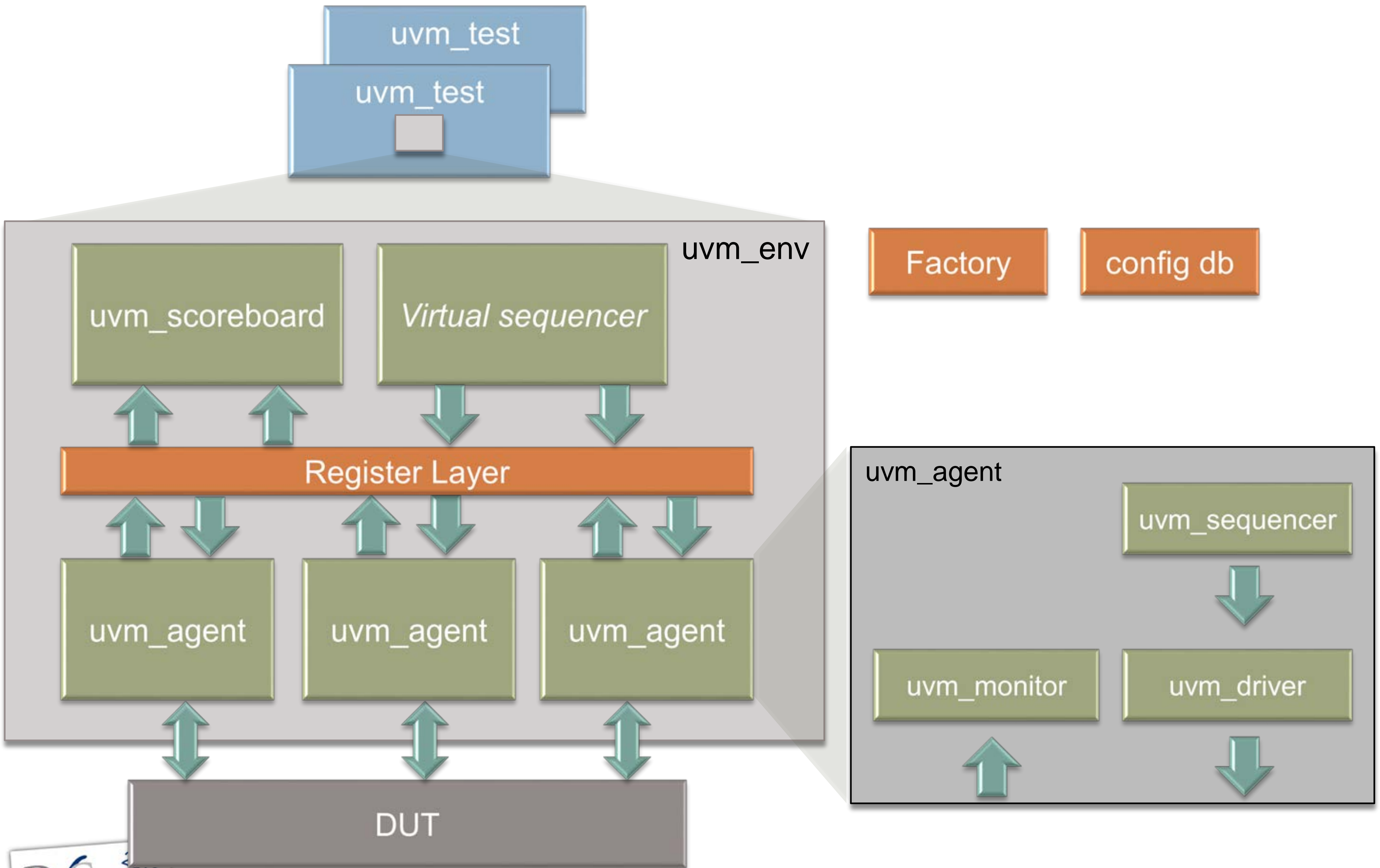
The Finer Points of UVM



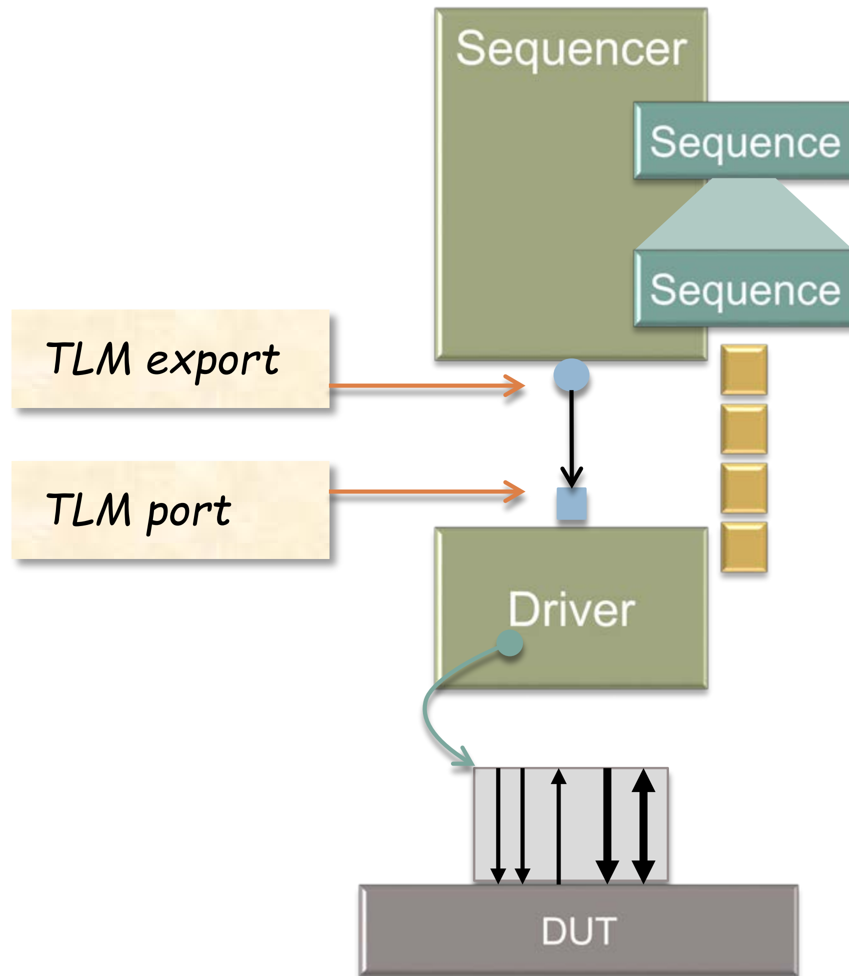
- Sequences and sequencers
- The arbitration queue
- Virtual sequences
- Request and response
- Multiple sequencer stacks

easier UVM

The Big Picture



Sequences and Sequencers




```
start_item(req);
finish_item(req);
```

```
seq_item_port.get(req);
```

DVCon 2013
Design & Verification Conference & Exhibition

A Simple Sequence

```
class my_seq extends uvm_sequence #(my_tx);
  `uvm_object_utils(my_seq)

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

  task body;
    repeat(4)
    begin
      req = my_tx::type_id::create("req");
      start_item(req);

      if (!req.randomize())
        `uvm_error("", "failed to randomize")

      finish_item(req);
    end
  endtask
endclass
```

DVCon 2013
Design & Verification Conference & Exhibition

Nested Sequences

```
class top_seq extends uvm_sequence #(my_tx);
  ...

  `uvm_declare_p_sequencer(my_seqr)
  ...

  task body;
    repeat(3)
    begin
      my_seq seq;
      seq = my_seq::type_id::create("seq");

      if (!seq.randomize())
        `uvm_error("", "failed to randomize")

      seq.start(p_sequencer, this);
    end
  endtask
  ...
```

Variable that points to sequencer

Sequencer

Parent sequence

DVCon 2013 Design & Verification Conference & Exhibition

Concurrent Sequences

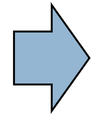
```
task body;
  fork
    begin
      seq1 = my_seq::type_id::create("seq1");
      if (!seq1.randomize())
        `uvm_error("", "failed to randomize")
      seq1.start(p_sequencer, this);
    end
    begin
      seq2 = my_seq::type_id::create("seq2");
      if (!seq2.randomize())
        ...
      seq2.start(p_sequencer, this);
    end
    begin
      ...
      seq3.start(p_sequencer, this);
    end
  join
endtask
```

Transactions will be strictly interleaved

DVCon 2013 Design & Verification Conference & Exhibition

The Finer Points of UVM

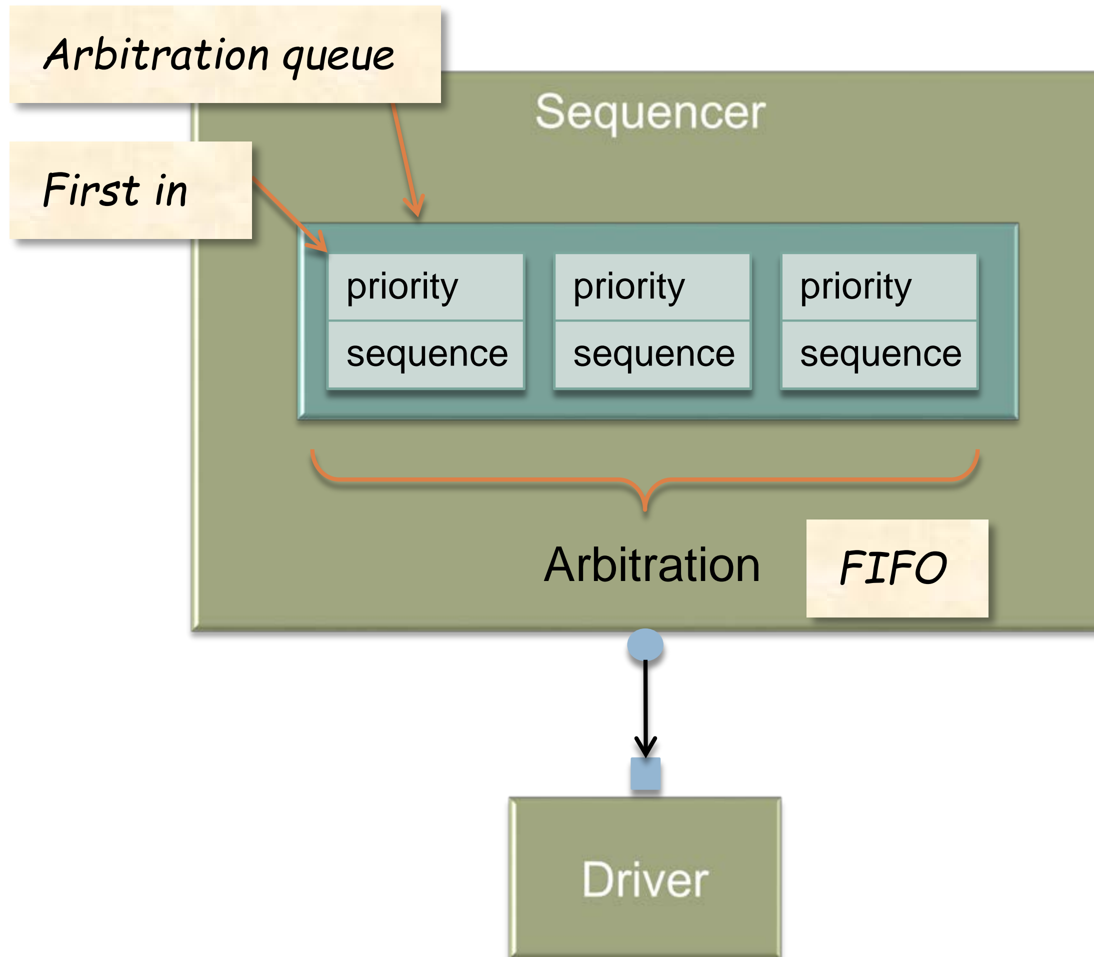
- Sequences and sequencers



- The arbitration queue
- Virtual sequences
- Request and response
- Multiple sequencer stacks

easier UVM

The Arbitration Queue




```
fork
   start
      body
         start_item
         finish_item

   start
      body
         start_item
         finish_item

   start
      body
         start_item
         finish_item
join
```

```
begin
   seq_item_port.get(req);

   seq_item_port.get(req);

   seq_item_port.get(req);
end
```

DVCon 2013 Design & Verification Conference & Exhibition

Setting the Arbitration Algorithm

```
task body;
  p_sequencer.set_arbitration(
                              SEQ_ARB_STRICT_RANDOM);
  fork
    begin
      seq1 = my_seq::type_id::create("seq1");
      if (!seq1.randomize())
        `uvm_error("", "failed to randomize")
      seq1.start(p_sequencer, this, 1);
    end
    begin
      ...
      seq2.start(p_sequencer, this, 2);
    end
    begin
      ...
      seq3.start(p_sequencer, this, 3);
    end
  join
endtask
```

Priority (default 100)

DVCon 2013 Design & Verification Conference & Exhibition

Arbitration Algorithms

Arbitration mode	Order in which requests granted
SEQ_ARB_FIFO	FIFO order (default)
SEQ_ARB_RANDOM	Random order
SEQ_ARB_STRICT_FIFO	Highest priority first, then FIFO order
SEQ_ARB_STRICT_RANDOM	Highest priority first, then random order
SEQ_ARB_WEIGHTED	Weighted by priority
SEQ_ARB_USER	User-defined

User-Defined Arbitration Algorithm

```
class my_sequencer extends uvm_sequencer #(my_tx);
  ...
  function integer user_priority_arbitration(
                                    integer avail_sequences[$]);
    foreach (avail_sequences[i])
    begin
      integer             index = avail_sequences[i];
      uvm_sequence_request req = arb_sequence_q[index];
      int                   pri = req.item_priority;
      uvm_sequence_base     seq = req.sequence_ptr;

      if (pri > max_pri)
        ...
    end
    return max_index;
  endfunction

endclass
```

Could access properties of the sequence object

DVCon 2013 Design & Verification Conference & Exhibition

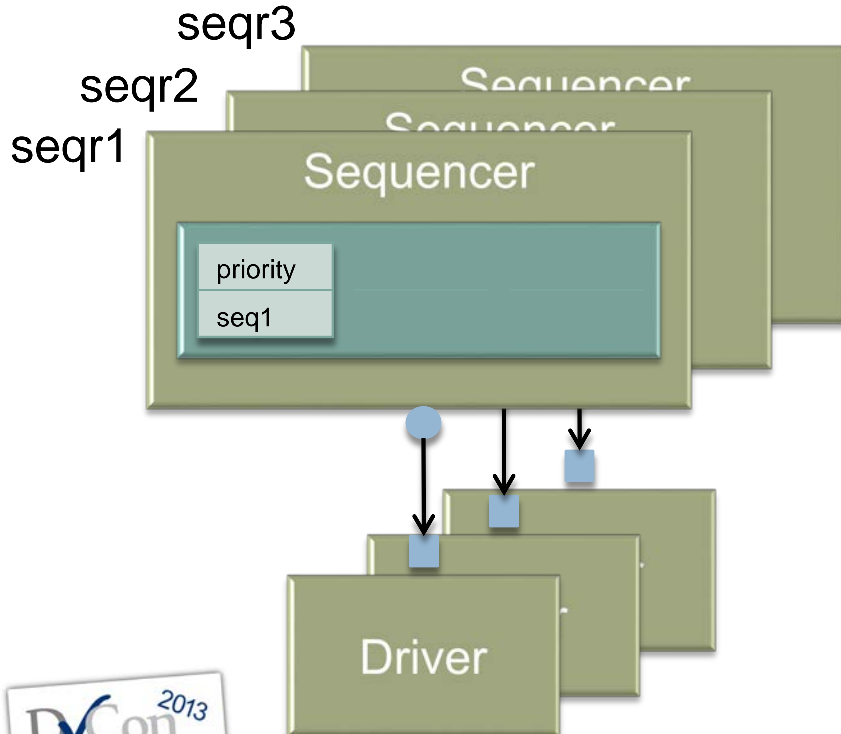
The Finer Points of UVM

- Sequences and sequencers
- The arbitration queue
- Virtual sequences
- Request and response
- Multiple sequencer stacks

easier UVM

Virtual Sequences

No transactions



Can be null

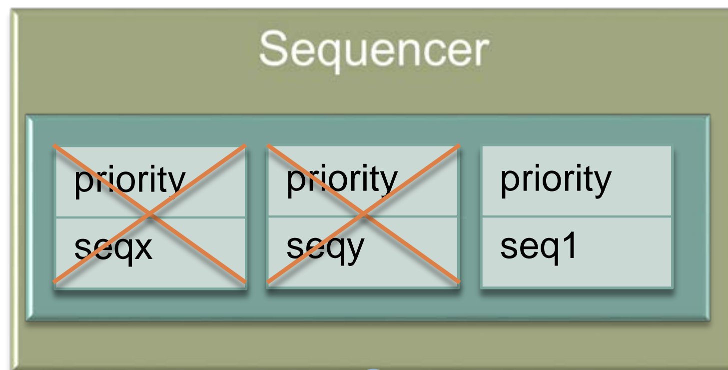
```
vseq.start(seqr0, null, priority)
  body
    fork
      seq1.start(seqr1, this)
        body
          start_item
          ...

      seq2.start(seqr2, this, 50)
        body
          start_item
          ...
```

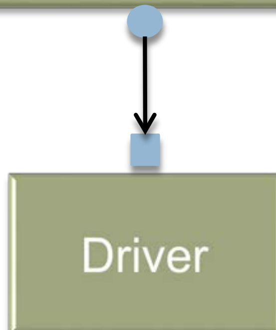
Blocks

Inherits priority

DVCon 2013 Design & Verification Conference & Exhibition

Sequencer Lock






```
vseq.start(seqr0, null)
  body
    begin
      this.lock(seqr1);
      seq1.start(seqr1, this);
        body
          start_item
          finish_item
      this.unlock(seqr1);
      ...
```

Important!

DVCon 2013

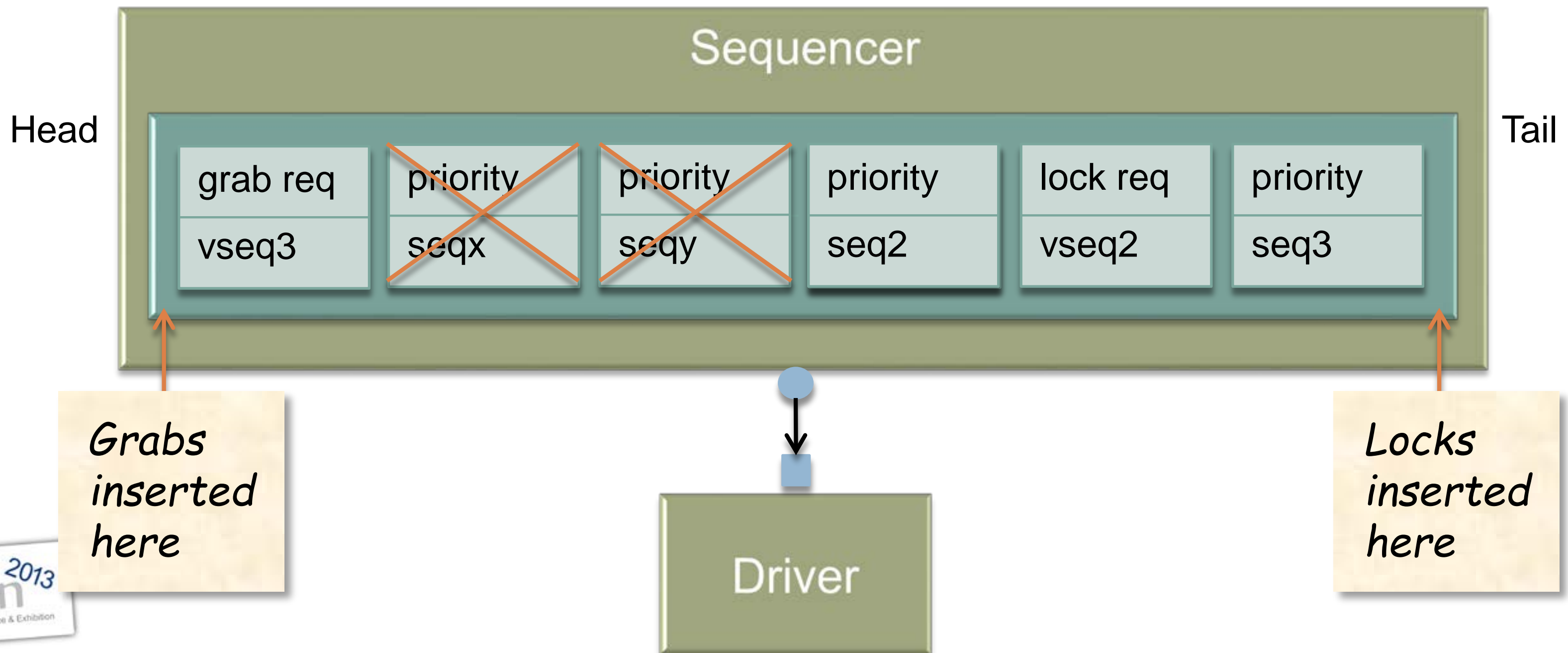
Lock versus Grab

Virtual sequence	Virtual sequence	Virtual sequence

```
vseq1.start
  body
    begin
      lock
        seq1.start
        unlock
```

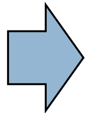
```
vseq2.start
  body
    begin
      lock
        seq2.start
        unlock
```

```
vseq3.start
  body
    begin
      grab
        seq3.start
        ungrab
```



The Finer Points of UVM

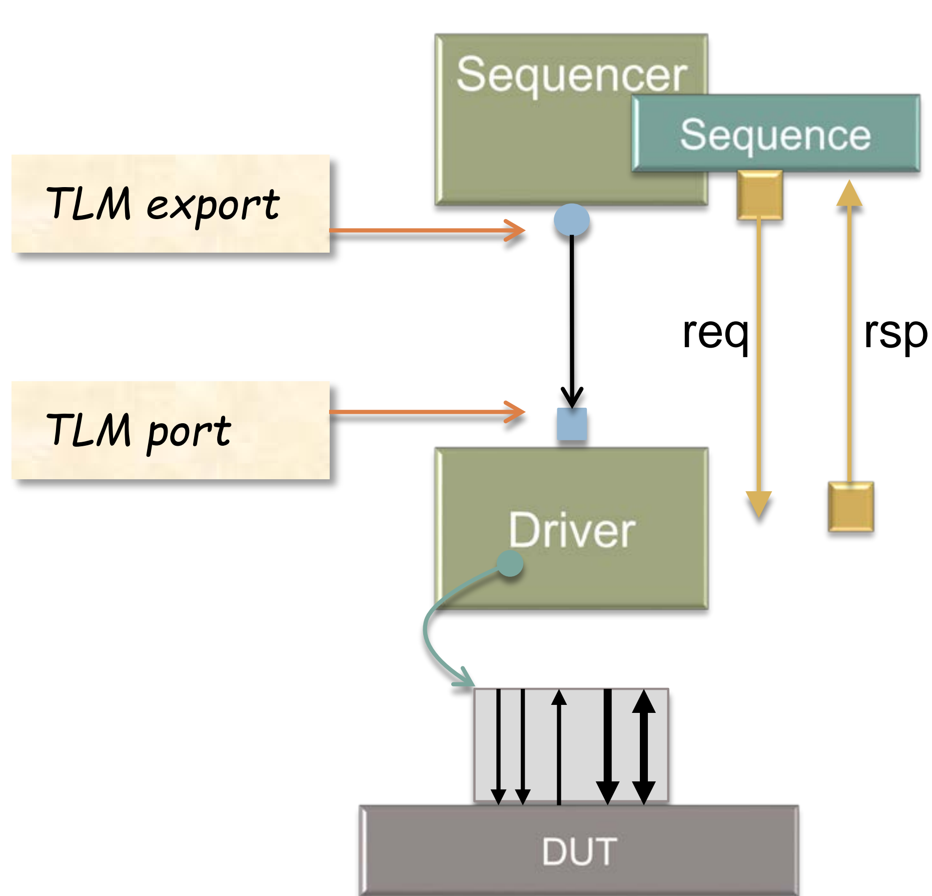
- Sequences and sequencers
- The arbitration queue
- Virtual sequences



- Request and response
- Multiple sequencer stacks

easier UVM

Request and Response

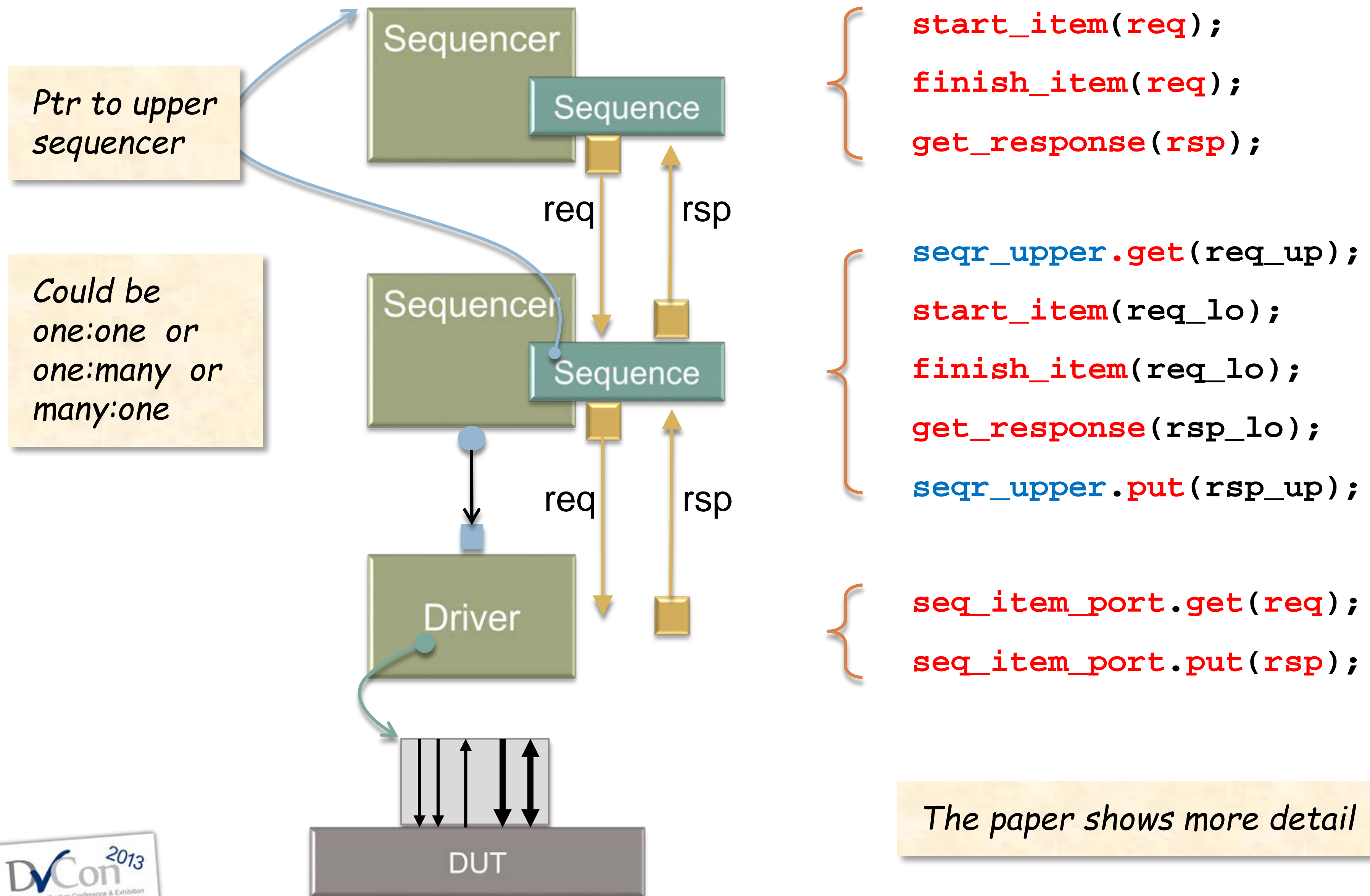



```
start_item(req);
finish_item(req);
get_response(rsp);
```

```
seq_item_port.get(req);
seq_item_port.put(rsp);
```

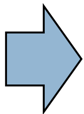
The paper describes in detail how to code pipelined req/rsp

Layered Sequencers



The Finer Points of UVM

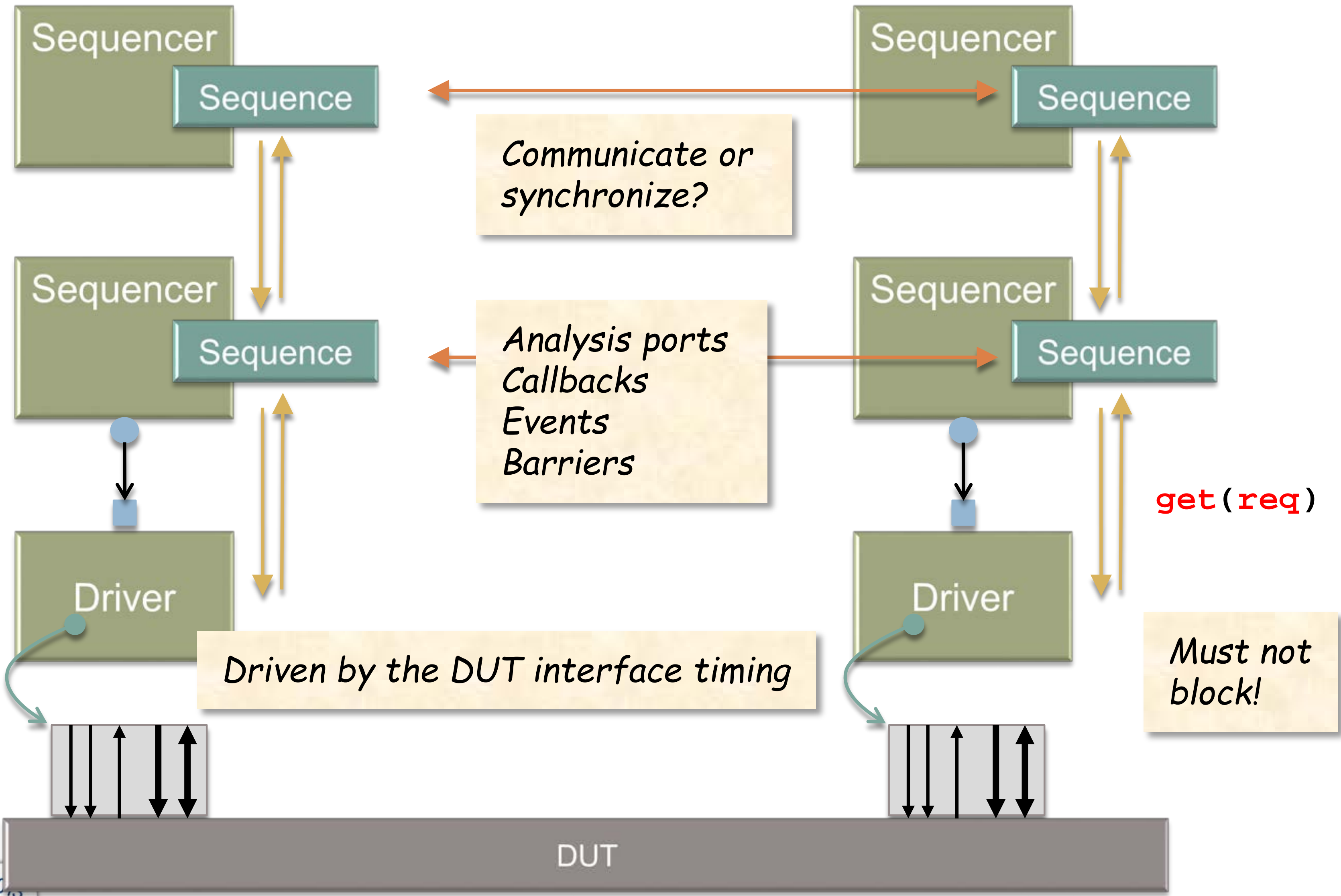
- Sequences and sequencers
- The arbitration queue
- Virtual sequences
- Request and response



- Multiple sequencer stacks

easier UVM

Multiple Agents / Sequencer Stacks



Driver calls try_next_item

```
seq_item_port.try_next_item(req);

if (req == null)
begin
  dut_vi.idle <= 1;
  ...
  @(posedge dut_vi.clock);
end
else
begin
  seq_item_port.item_done();

  dut_vi.idle <= 0;
  ...
  @(posedge dut_vi.clock);
  ...
  seq_item_port.put(rsp);
```

Wiggle pins for idle cycle

Must be called in same time step

Response can be pipelined

DVCon 2013 Design & Verification Conference & Exhibition

The Finer Points of UVM

Also in the paper

- The UVM sequence library
- Pipelined requests and responses
- The response handler
- UVM events and event pools
- The configuration database

easier UVM