Stateful Transaction and Automatic UVM Testbench Generation

Jie Ding
Qualcomm Inc.

August 12th, 2014
SNUG Boston

Agenda

Stateful Sequence Item

Application Scenarios

Automatic UVM generation

Conclusion

Stateful Sequence Item

Parallel or Sequencial?

Sequence/agent interaction recount

Stateful Sequence Item

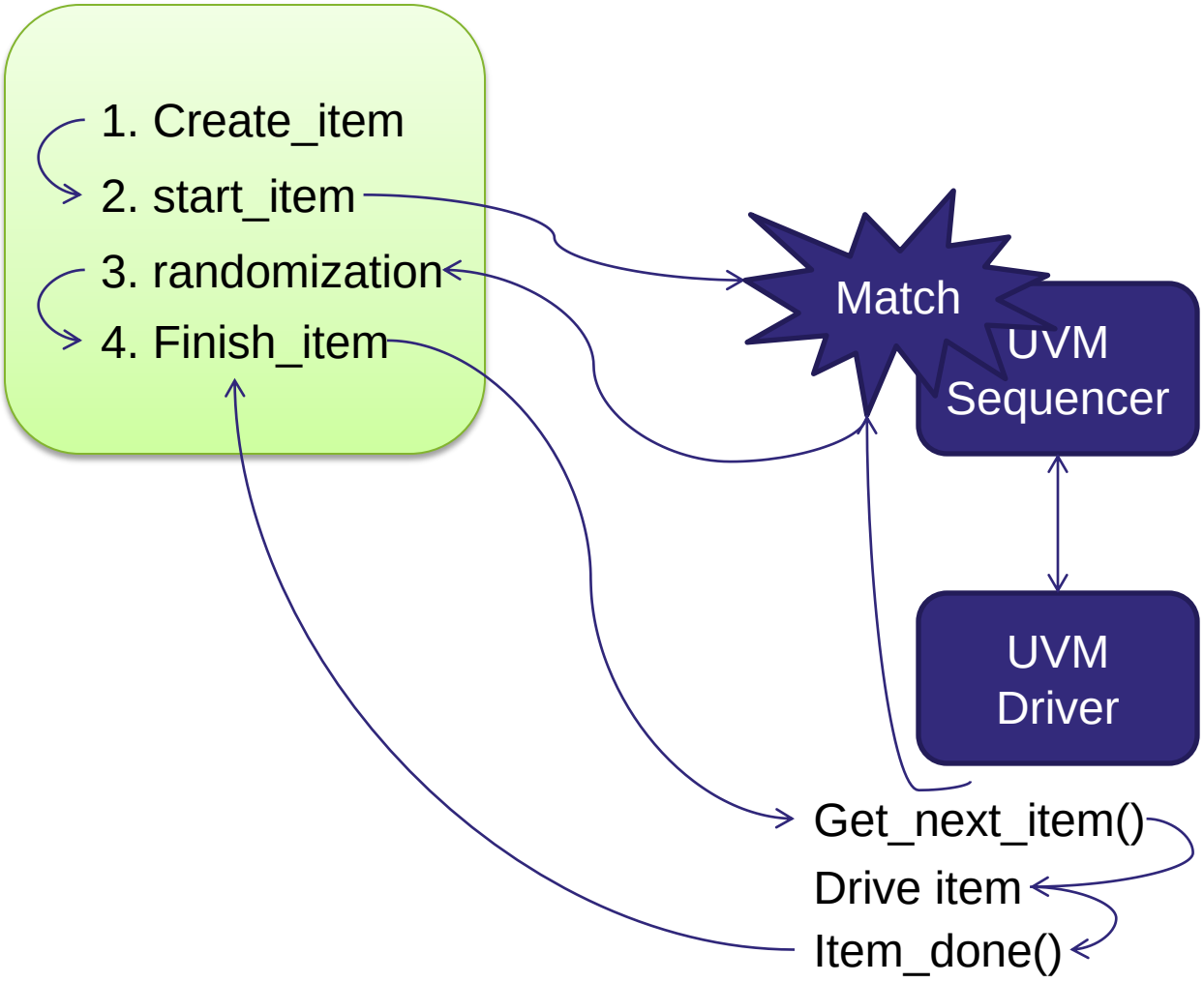
Parallel or Sequencial

- UVM_component (parallel), UVM_sequence (sequencial)

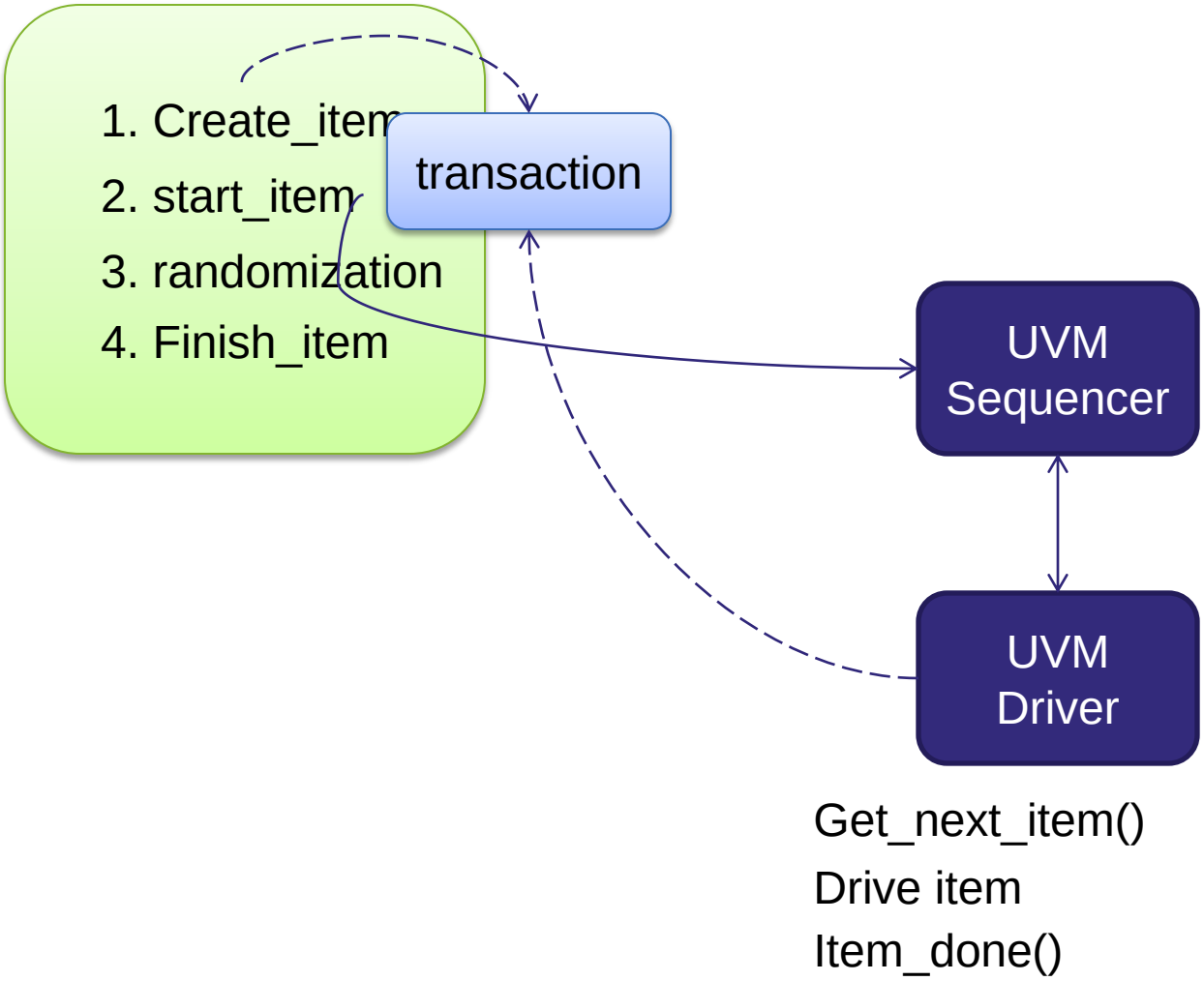
- Two module level environment style
  - Minimum agent, complexity in sequence library
    - Top level State machine
    - Handshaking protocols
  - Minimum sequence, complexity in agents and checkers
    - Out of order engine.
    - Bus pipleline
    - SOC environments

Simple and easy to use communicate method is key.

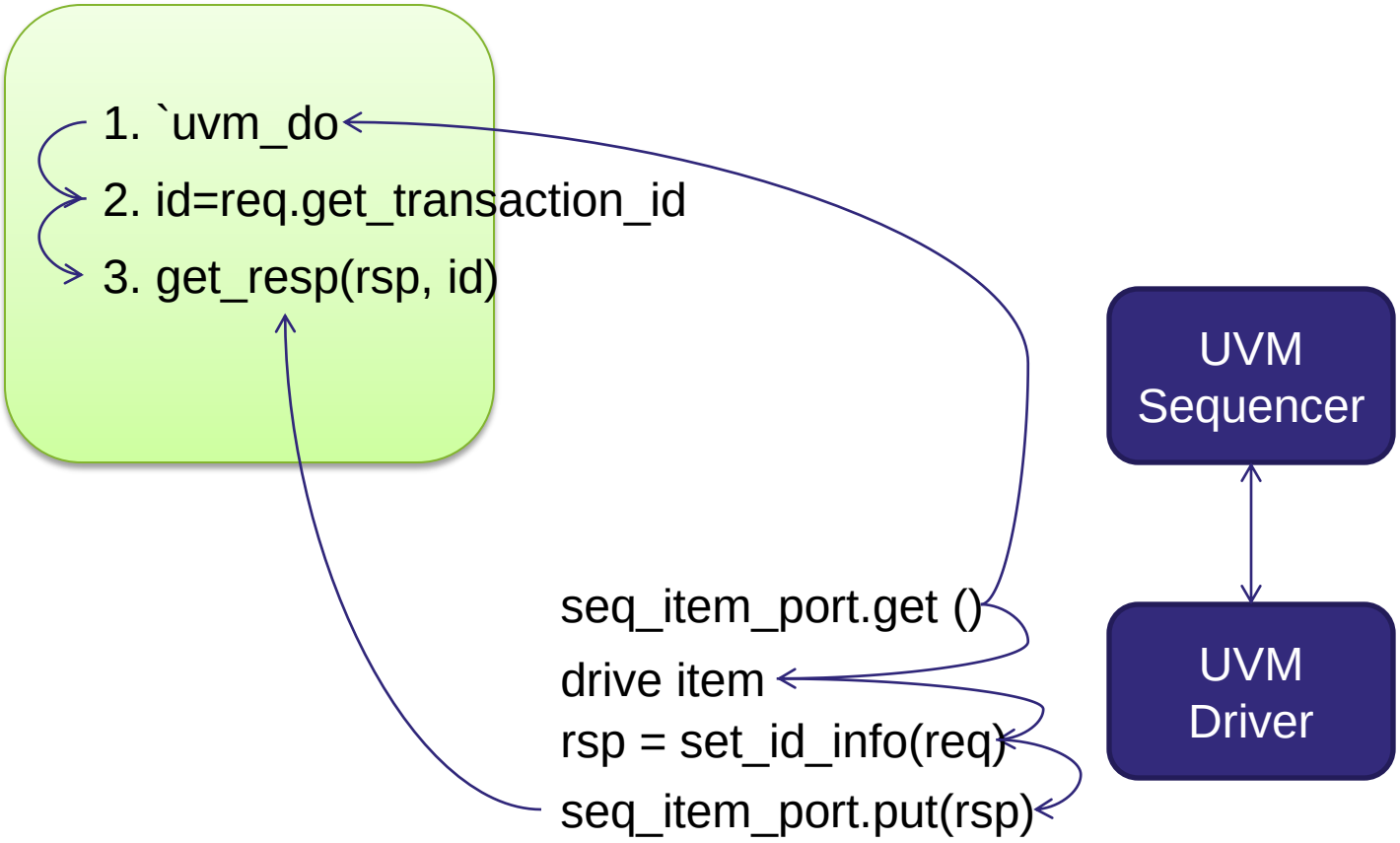
UVM Agent Control Flow



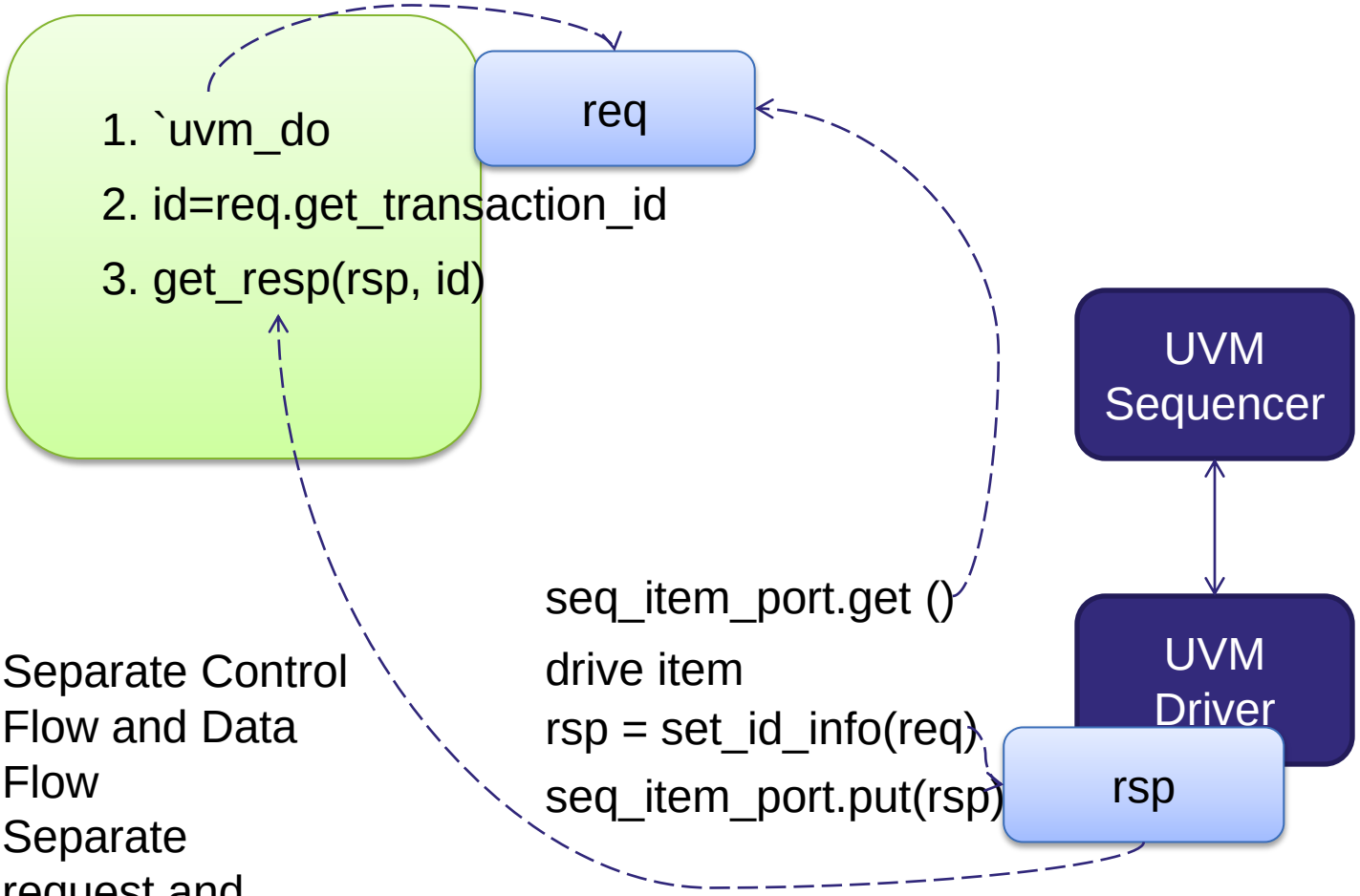
UVM Agent Data Flow



UVM Agent Response Control Flow

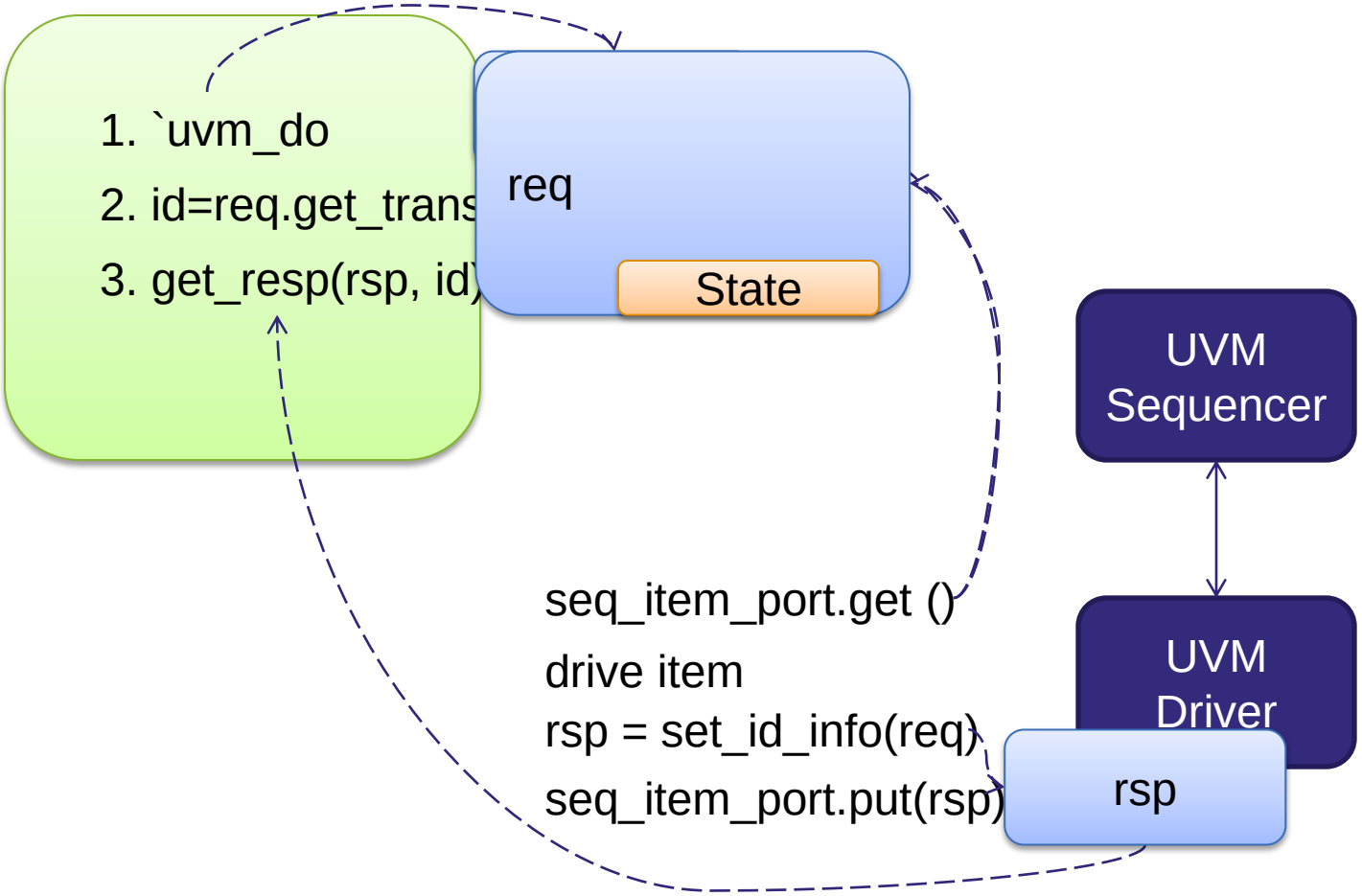


UVM Agent Response Data Flow



1. Separate Control Flow and Data Flow
2. Separate request and response
3. ID based synchronization

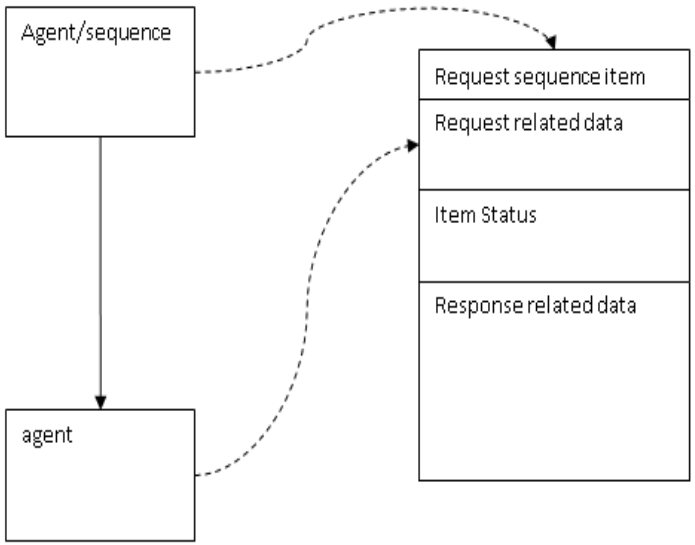
Stateful Transaction



Stateful Transaction

- Response inside request transaction
  - No ID needed to match request and response
  - Request data, response data strictly separate
- State is added for synchronization
  - Request/response data update BE atomic
  - Data update always follow by state update

Stateful Sequence Item (inside)




```
class sis_base extends uvm_object;
…
local string status;
event       status_change;
…
task tb_status::set_as(string status);
      process_sem.get(); // Don't remove this, this is needed to resolve race
                         // condition
#0;                      // Wait a delta time for all other wait to get triggered.
      this.status = status;
      -> status_change;
      process_sem.put();
endtask

task tb_status::wait_for(string status, time timeout = 0);
      while(this.status != status) begin
            @(status_change);
      end
endtask
…
endclass
```

Howto Use it

- Instantiate “tb_status” in your sequence_item or sequence.

```
class my_sequence_item …
        tb_status status = new();
        …
endclass
```

And you get

- status.wait_for(string status);
  - Block current thread until status match
- status.set_as(string status);
  - Set status to a string
  - Unblock any waiting components
- And other functions
  - get(), wait_for_list(string status[$]), etc.

Stateful Transaction

- Is it global data?
  - Access control is available
  - Created when needed
  - Destructed after use
- Easy Debug
  - Follow the state change and data change in transaction

Usage Scenarios

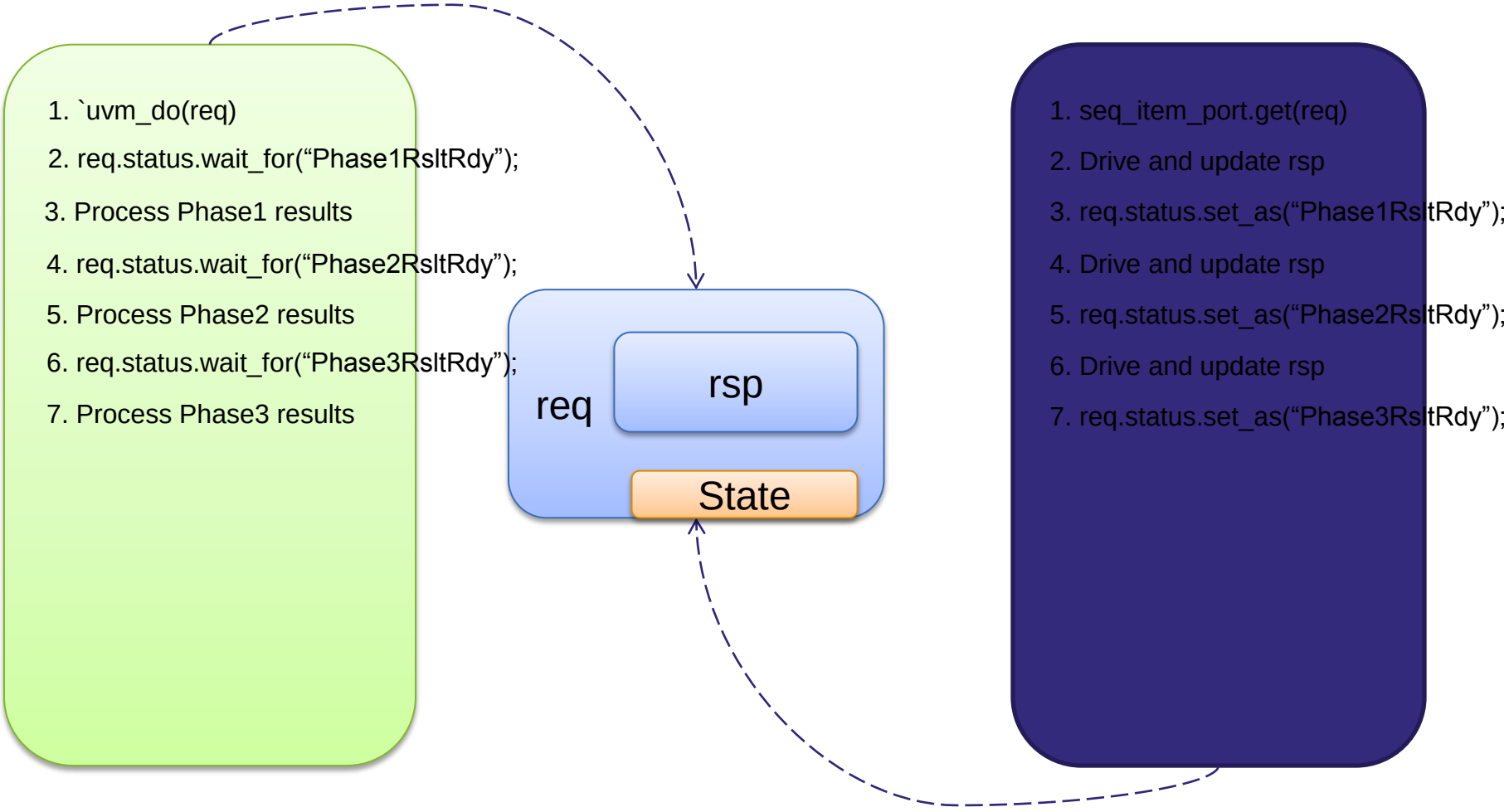
Multiphase scenario

Out of Order responses

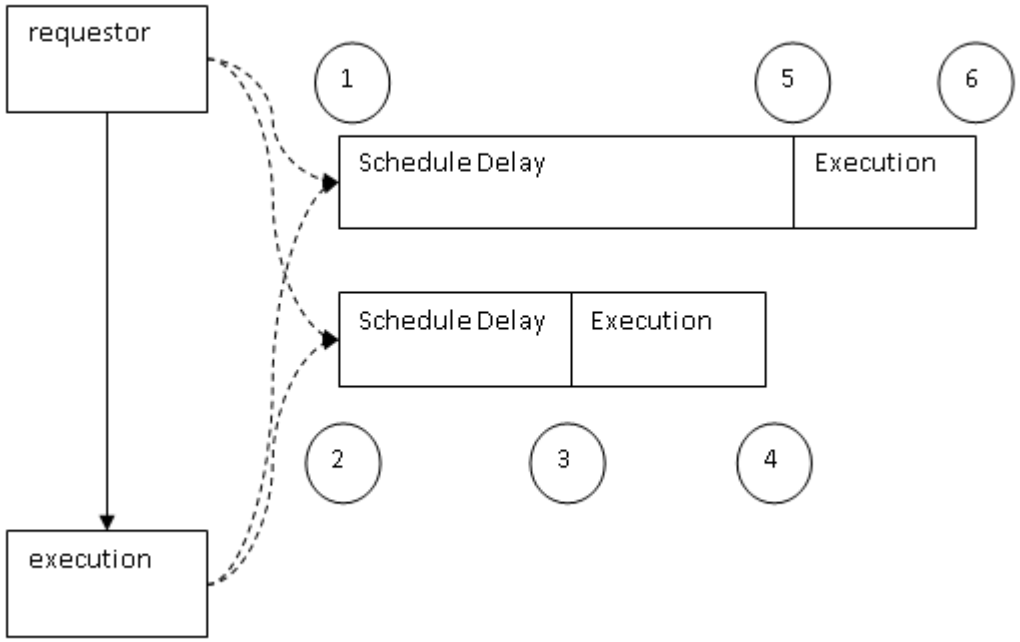
Multi-component scenario

Multi-component, multi-phase, out-of-order scenario

Stateful Transaction-Multiple Phase

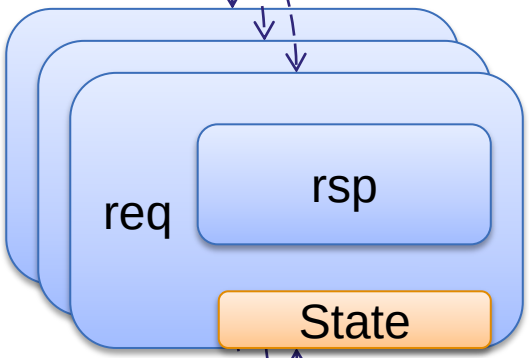


Out-of-order Scenario



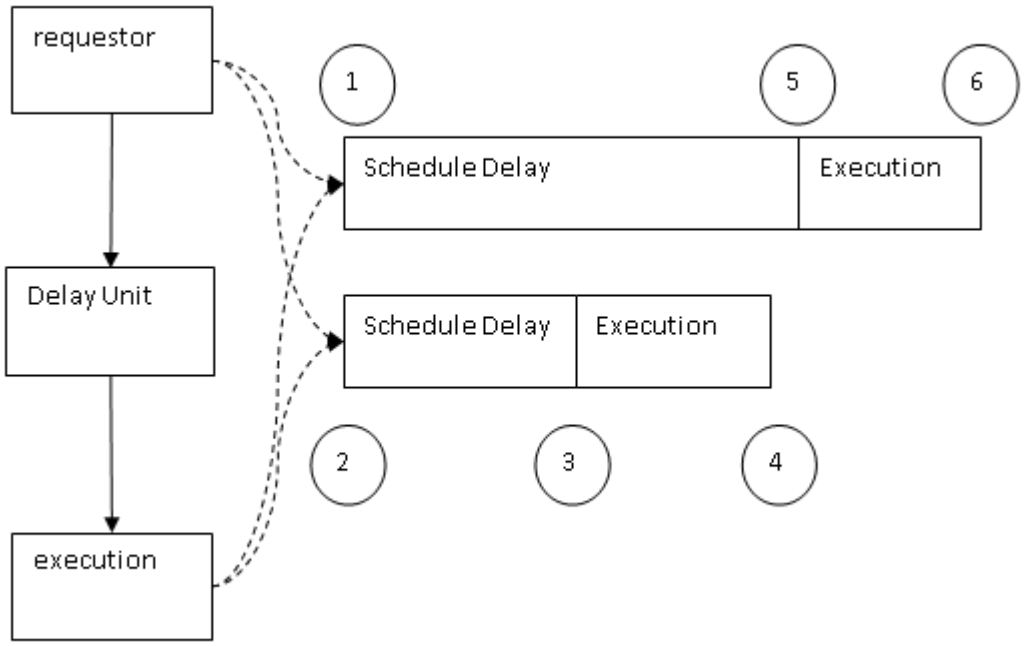
Stateful Transaction-Multiple Request

```
1. `uvm_do(req1)
2. `uvm_do(req2)
3. `uvm_do(req3)
4. fork
     begin
       req1.status.wait_for("DONE");
       process resp1;
     end
     begin
       req2.status.wait_for("DONE");
       process resp2;
     end
     begin
       req3.status.wait_for("DONE");
       process resp3;
     end
   join_all
```

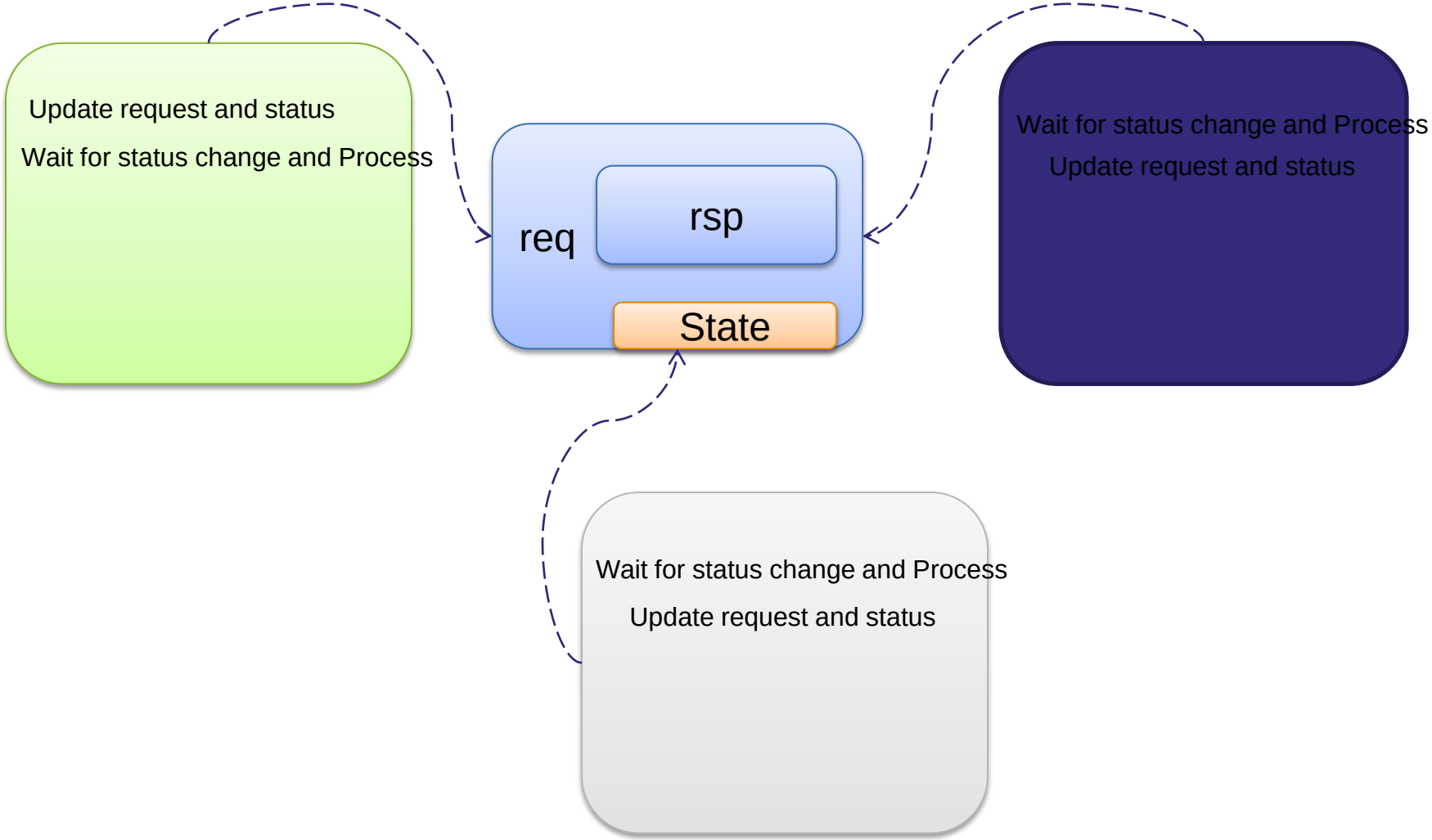



```
forever begin
   seq_item_port.get(req);

   //process request
   fork begin
      req_tmp = req;
      get_rsp(req_tmp.rsp);
      req_tmp.status.set_as("DONE");
   join_none

end
```

Multiple component out-of-order



Stateful Transaction-Multiple Components



Automatic UVM generation

Stateful Sequence Item Based

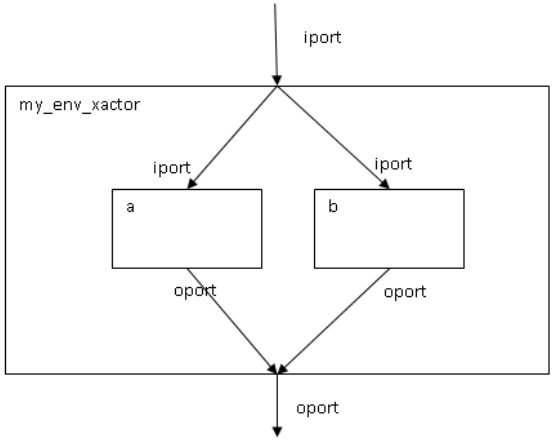
Building Block

- Basic Building Block is a transactor
  - Multiple input ports
    - Each input is a sequencer containing a TLM fifo
    - Sequencer is for sequence
    - TLM is for component
  - Multiple output ports
    - Each output port is UVM port to TLM fifo
  - Derived from UVM_driver
  - Can contain other transactors
- Sequence Item is Transaction
  - Status automatically added

Operators

- '->' connect operators
  - Connect input ports to output ports
  - Broadcasting from input port to all output ports
  - Unicast function available
- '=>' replace operators
  - Will override internal component
- User code are 'included' into generated coded
  - EDA compiler can locate user source code
- XF language compiler will generate all UVM framework

Xf file example




```
xactor my_xactor( <input_port_type iport>  ->  <output_port_type oport>)
{
        [code “my_xactor.sv”];
}

xactor my_env_xactor( <input_port_type iport>  ->  <output_port_type oport>)
{
        my_xactor   a, b;
        iport -> a.iport;
        iport -> b.iport;
        a.oport -> oport;
        b.oport -> oport;
        [code “my_xactor.sv”];
}
```

More information will be freely available.

Conclusion

- Propose a new way of communication
  - Combine request and response
  - Add status to sequence item(transaction)
- Created a quick prototyping model and language
  - No more manual testbench construction, focus on function.

COMPANY LOGO

snug
Synopsys Users Group

Thank You