User Experience with UVM

Stephen D'Onofrio & Peter D'Antonio

Stacking Verification Components in UVM

Approved for Public Release: 12-0309
Distribution Unlimited

Overview

- Background
- UVM Motivation
- Reuse Requirements
- Stacking UVCs Requirements & Architecture
- Code Examples
- Conclusion

Background

- **Verification at MITRE**
  - Effective verification flow for DSP combines analysis, system modeling and RTL verification
  - Worked well in previous technologies
  - File based, directed tests; capacity does not scale with technology
- **Project need**
  - 45nm ASIC, DSP datapath design
  - Schedule/resource constrained, 1st pass success
  - Step function verification flow improvement needed to meet increased design complexity

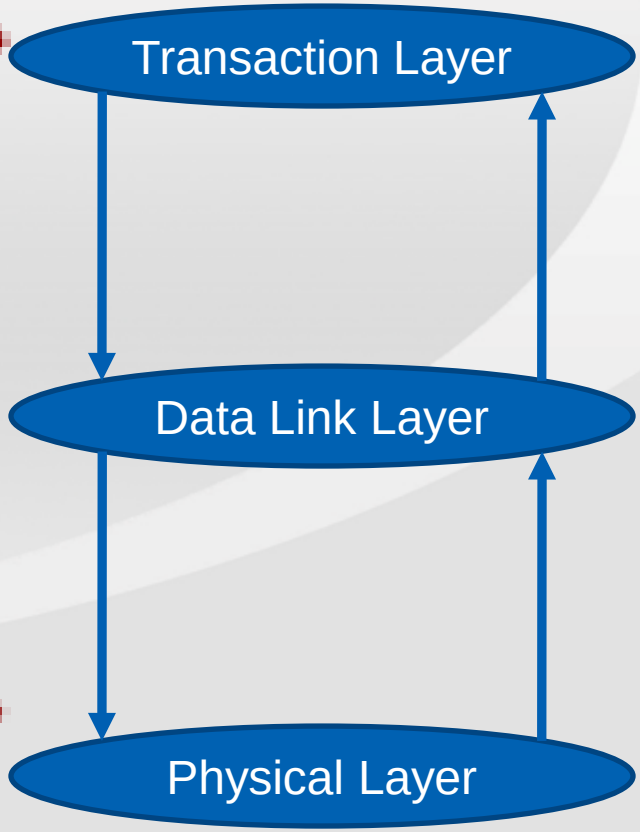
UVM Motivation

- **Quality**
  - Improve functional verification and ultimately product
- **Efficiency**
  - Return on investment through reuse
  - Recommended structure gives us a foundation to build
- **Leverage**
  - Broad industry support
  - User momentum
  - EDA vendor independent

Reuse Requirements

- **Reuse as much as possible!**
- **UVCs**
  - Configurable behavior
- **DUT Expect models**
  - Reuse between unit and system
  - Components with TLM ports
- **Sequences**
  - Abstract UVCs
  - Virtual sequences

Reuse Problem

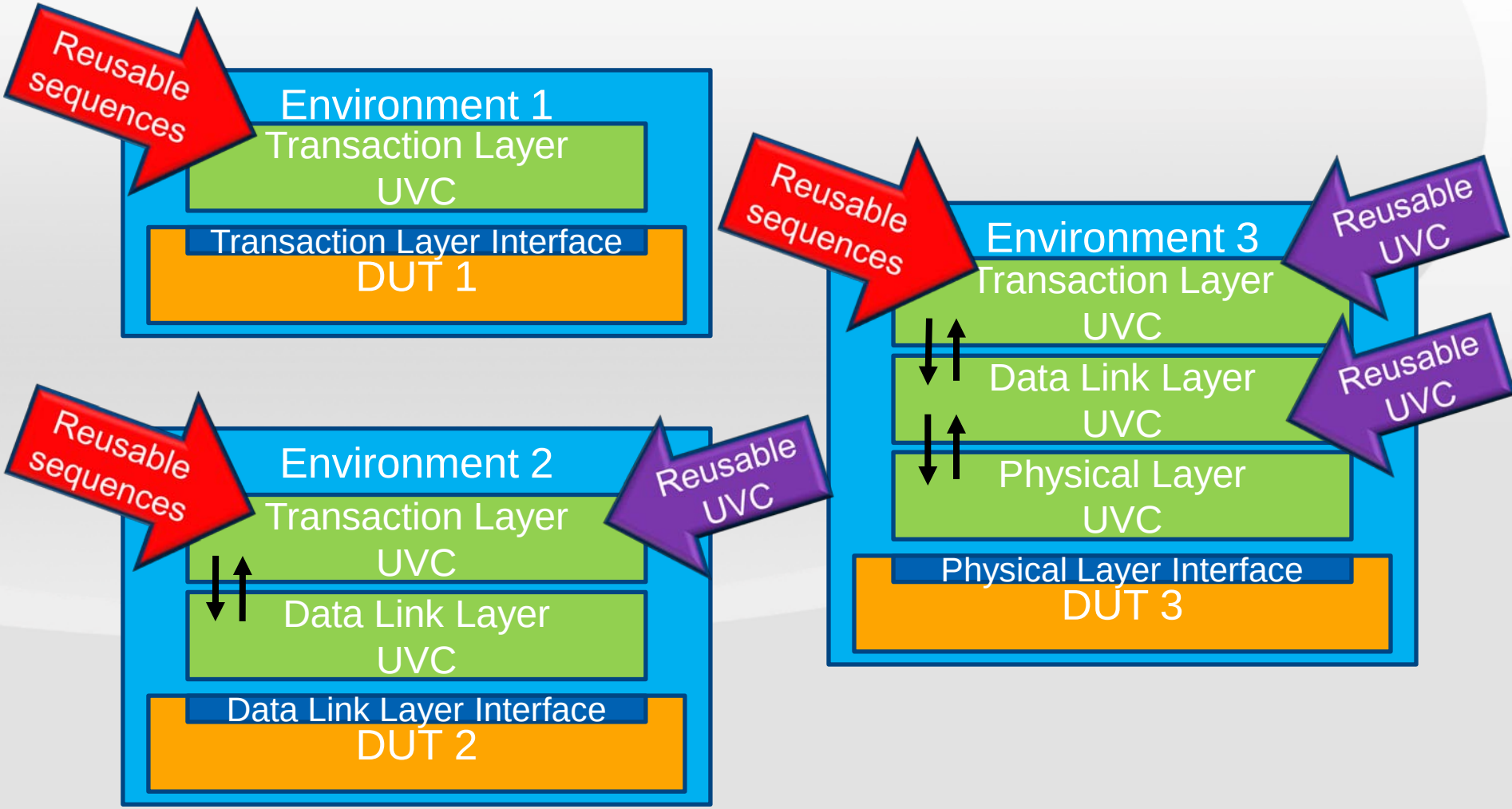


DVCon 2012 Design & Verification Conference & Exhibition

UVM

accellera SYSTEMS INITIATIVE

Vertical Reuse Solution

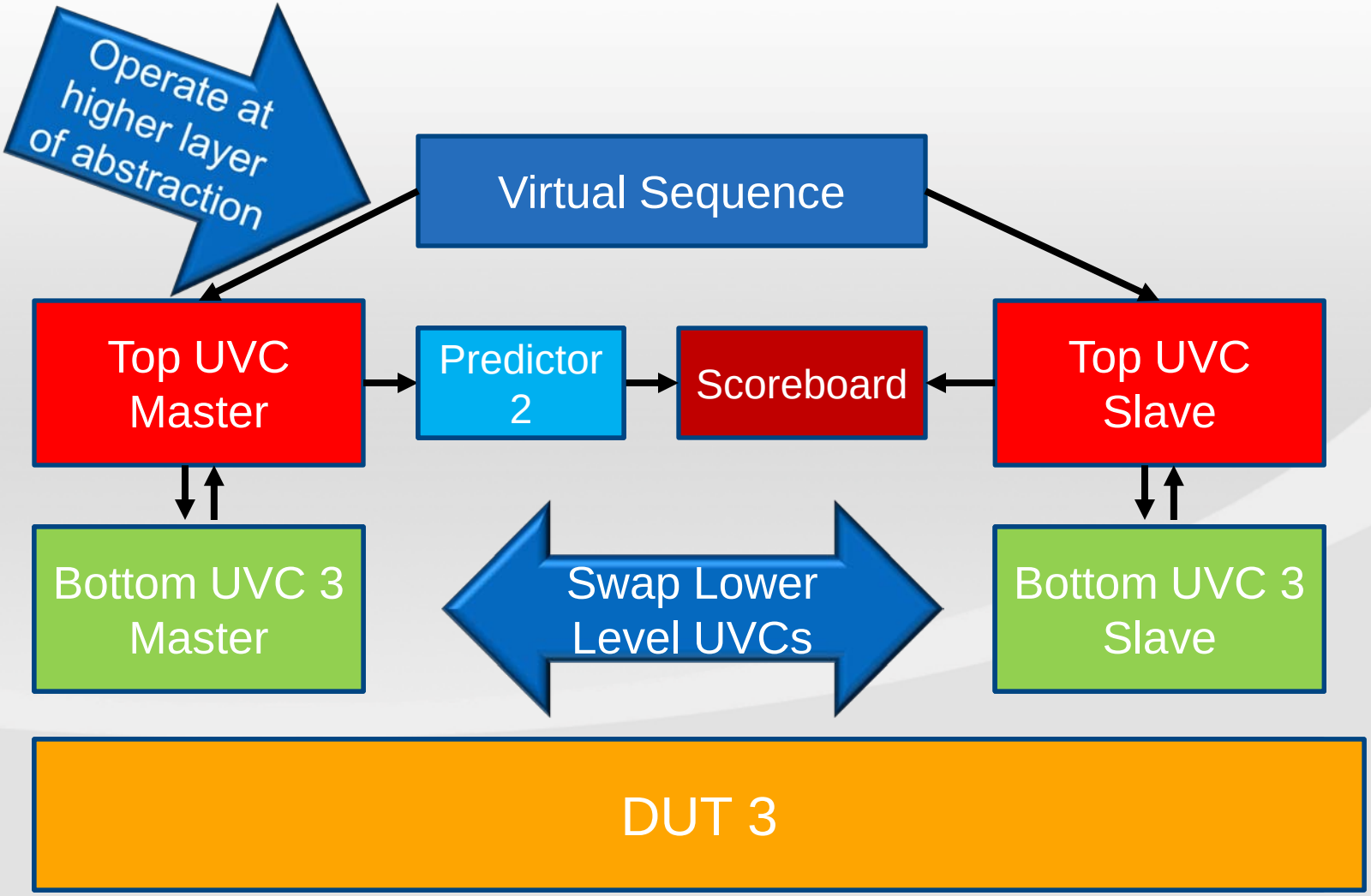


DVCon 2012 Design & Verification Conference & Exhibition

UVM

accellera SYSTEMS INITIATIVE

Horizontal Reuse Solution



Overview

- Background
- UVM Motivation
- Reuse Requirements
- **Stacking UVCs Requirements & Architecture**
- **Code Examples**
- Conclusion

UVM Layering Shortcomings

- User Guide missing layering monitor path
- No methodology for layering across UVCs
- No UVCs (agents) TLM interconnection strategy

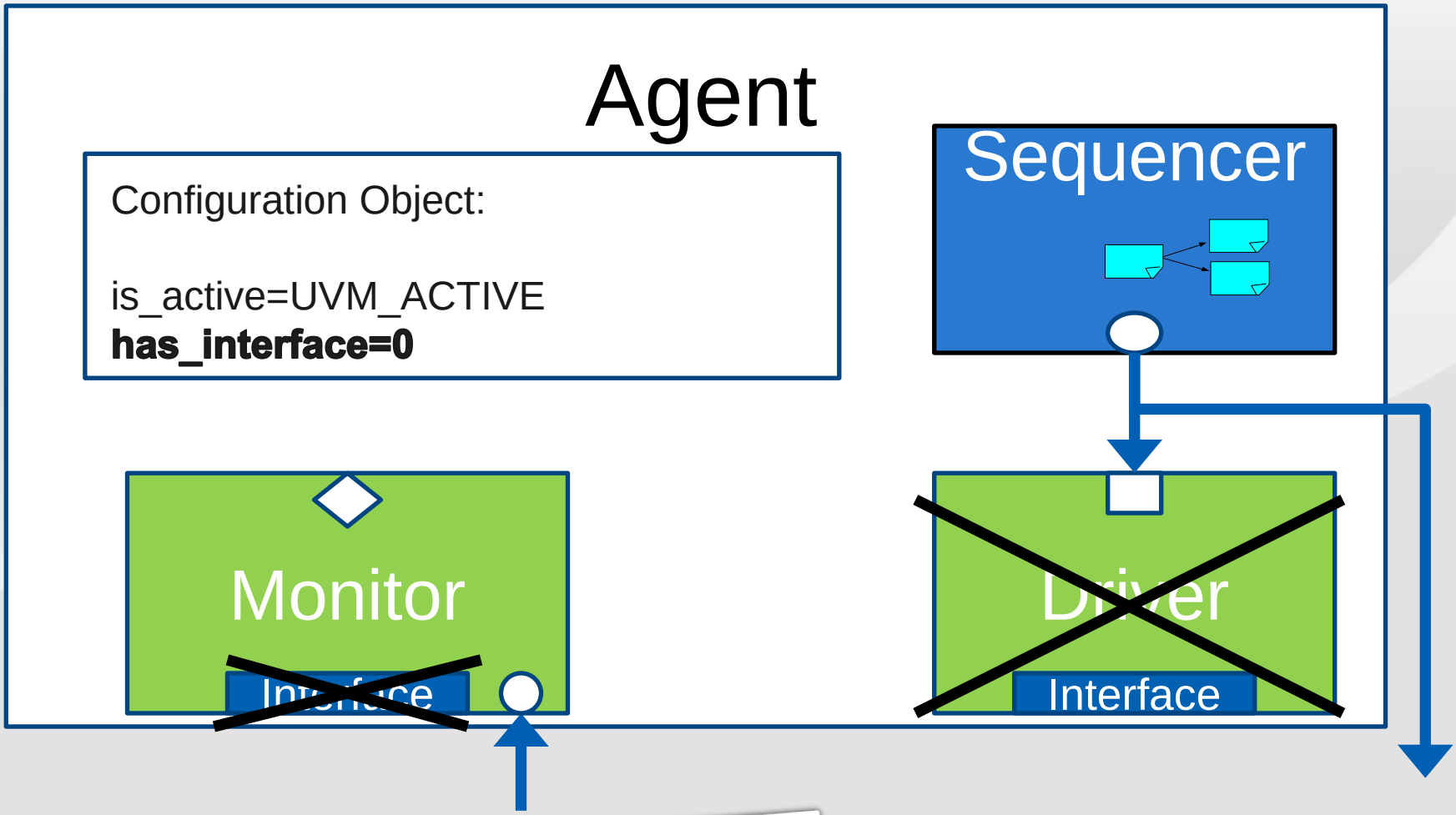
What is stacking UVCs?

- A framework for UVCs to handle layering
- Connecting or stacking one or more UVCs
- Easily swap UVCs anywhere in a stack
- Arbitrary UVC to UVC connection

Stacking Requirements

- **UVCs**
  - Optionally include interface
  - No knowledge of other UVCs
- **Adapter package**
  - **Converts** transactions
  - Connects 2 or more stacked UVCs together
  - Must be reusable
    - Operate in both Active/Passive modes
    - Includes a configuration object

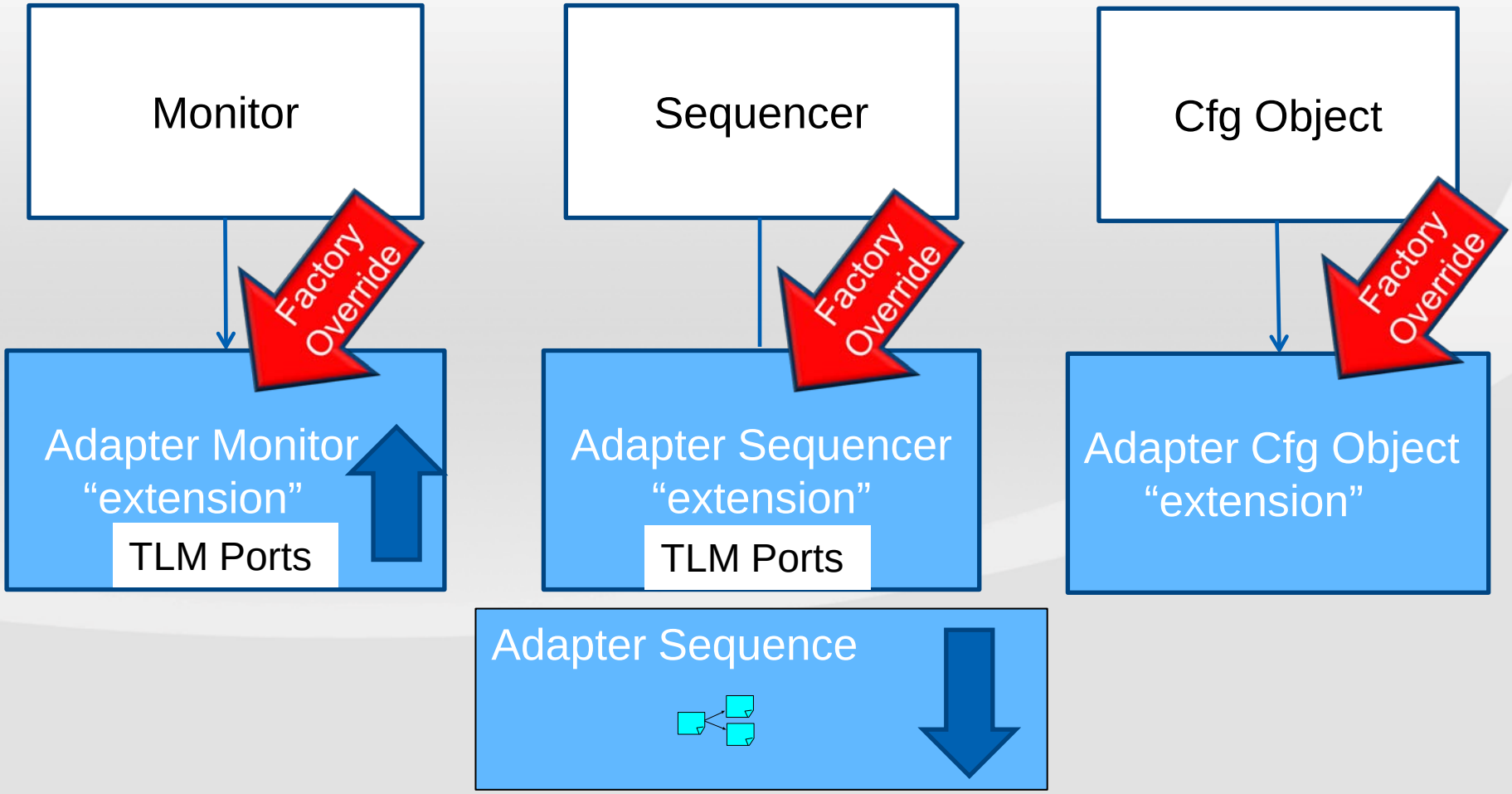
UVC



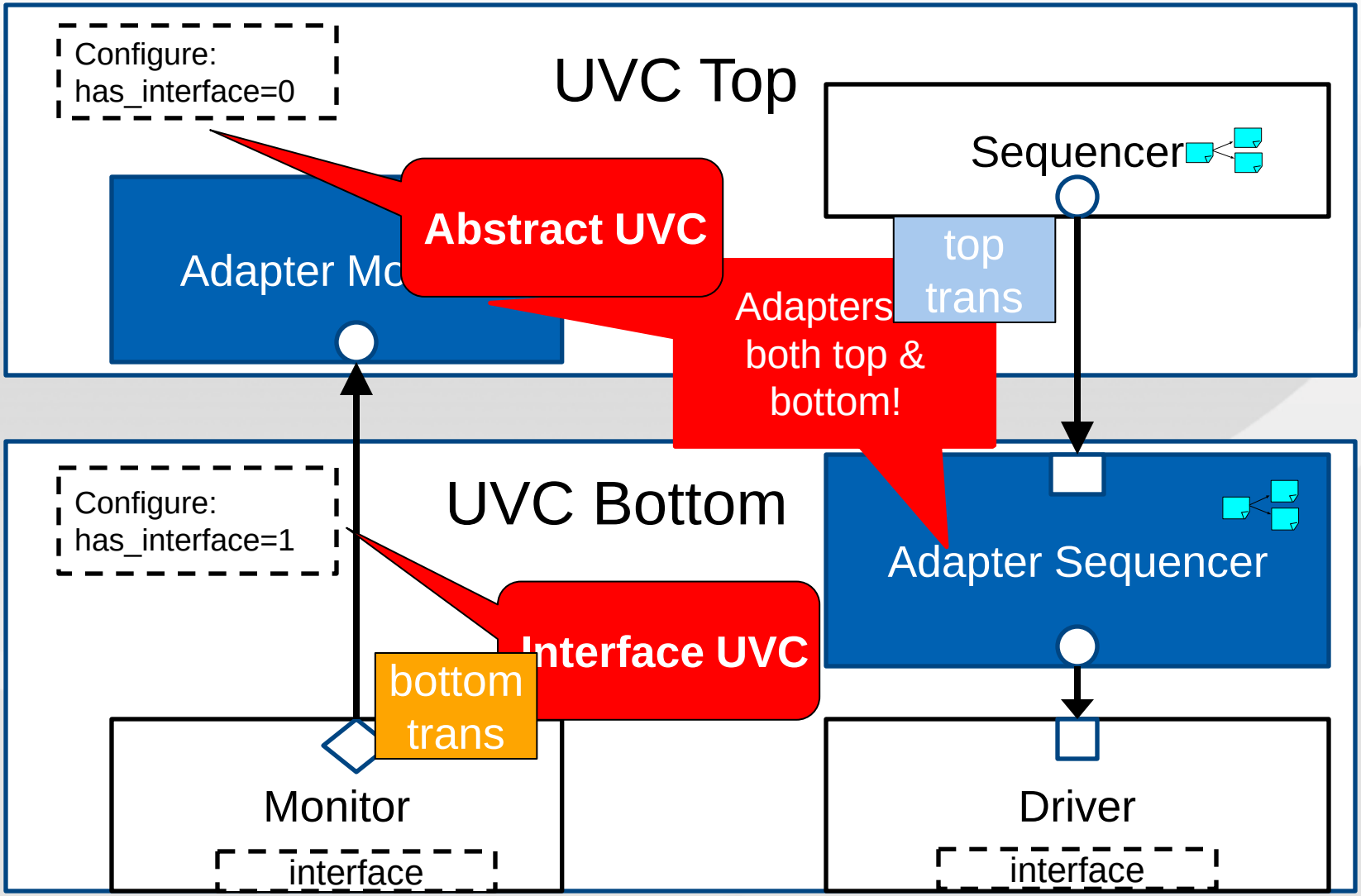
UVC Agent Code Snippet

```
class top_agent extends uvm_agent;

  bit has_interface = 1;

  function void build_phase(uvm_phase phase);
   void'(uvm_config_db#(bit)::get(this, "", "has_interface", has_interface));

    monitor = top_in_monitor::type_id::create("monitor",this);
    if(get_is_active() == UVM_ACTIVE) begin
      sequencer = top_in_sequencer::type_id::create("sequencer",this);
      if (has_interface)
        driver = top_in_driver::type_id::create("driver",this);
    end
  endfunction

  function void connect_phase(uvm_phase phase);
    if(get_is_active() == UVM_ACTIVE && has_interface)
      driver.seq_item_port.connect(sequencer.seq_item_export);
  endfunction
```

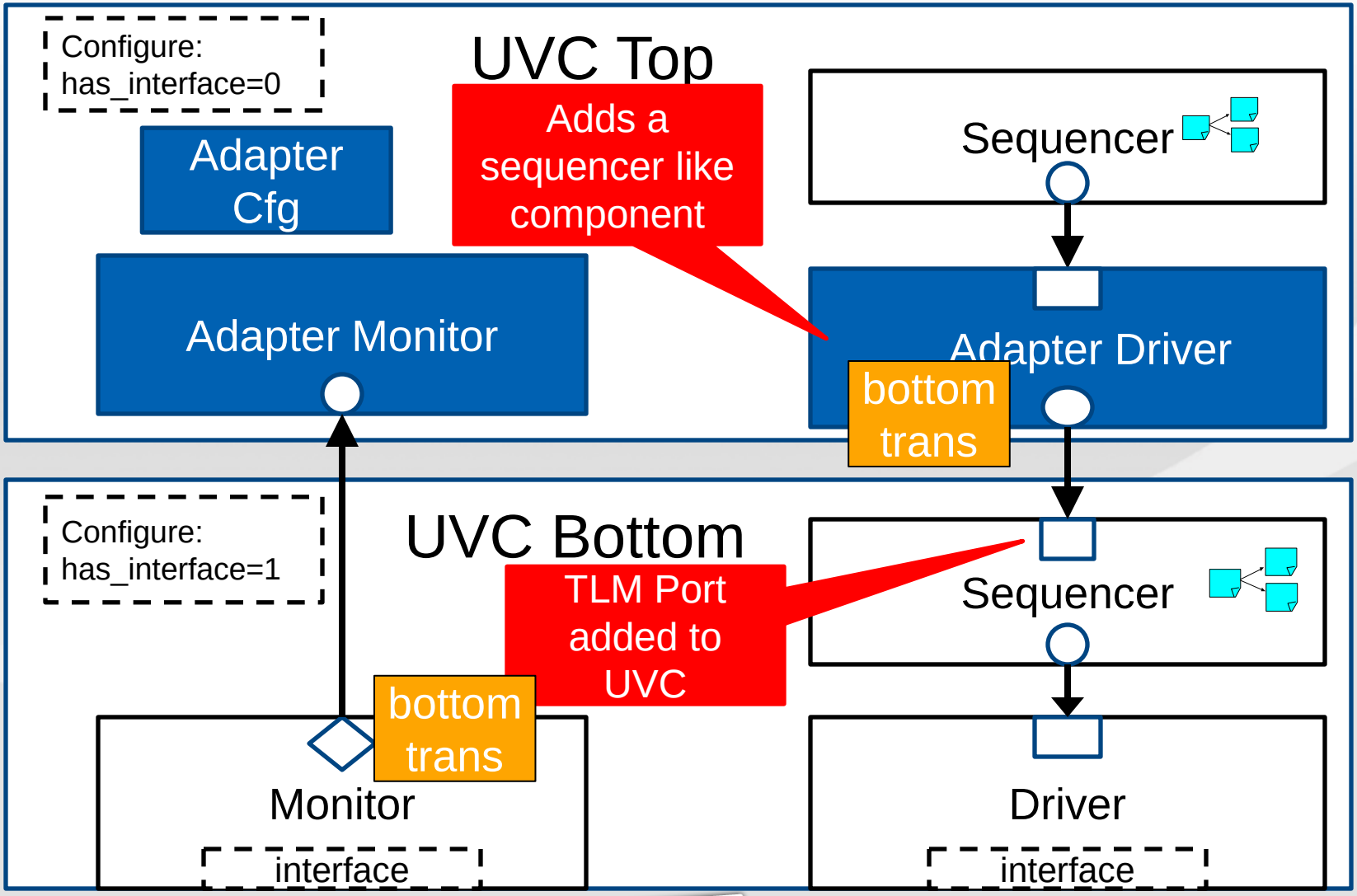
Adapter Package



Implementation Option A



Implementation Option B

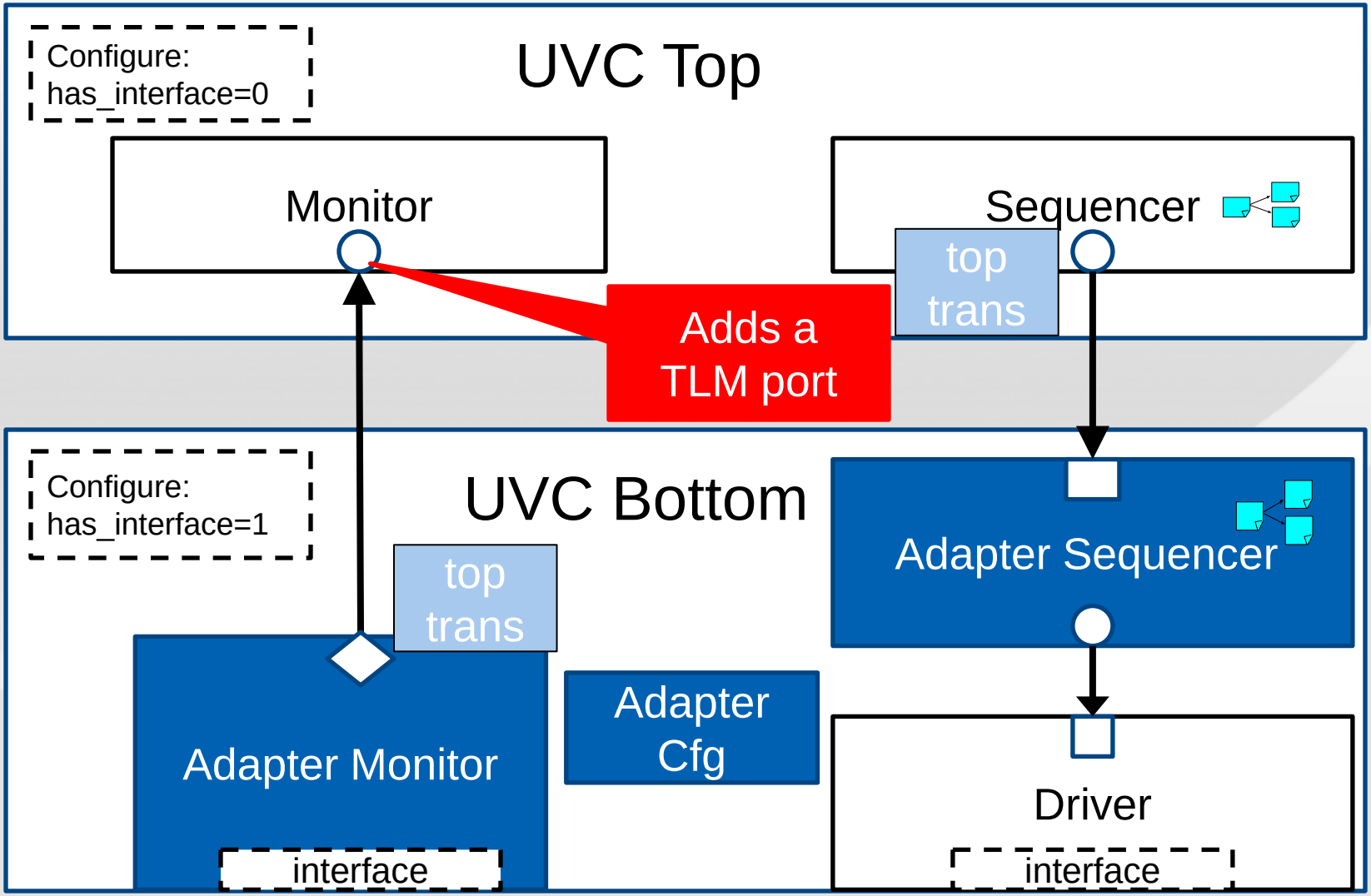


DVCon 2012 Design & Verification Conference & Exhibition

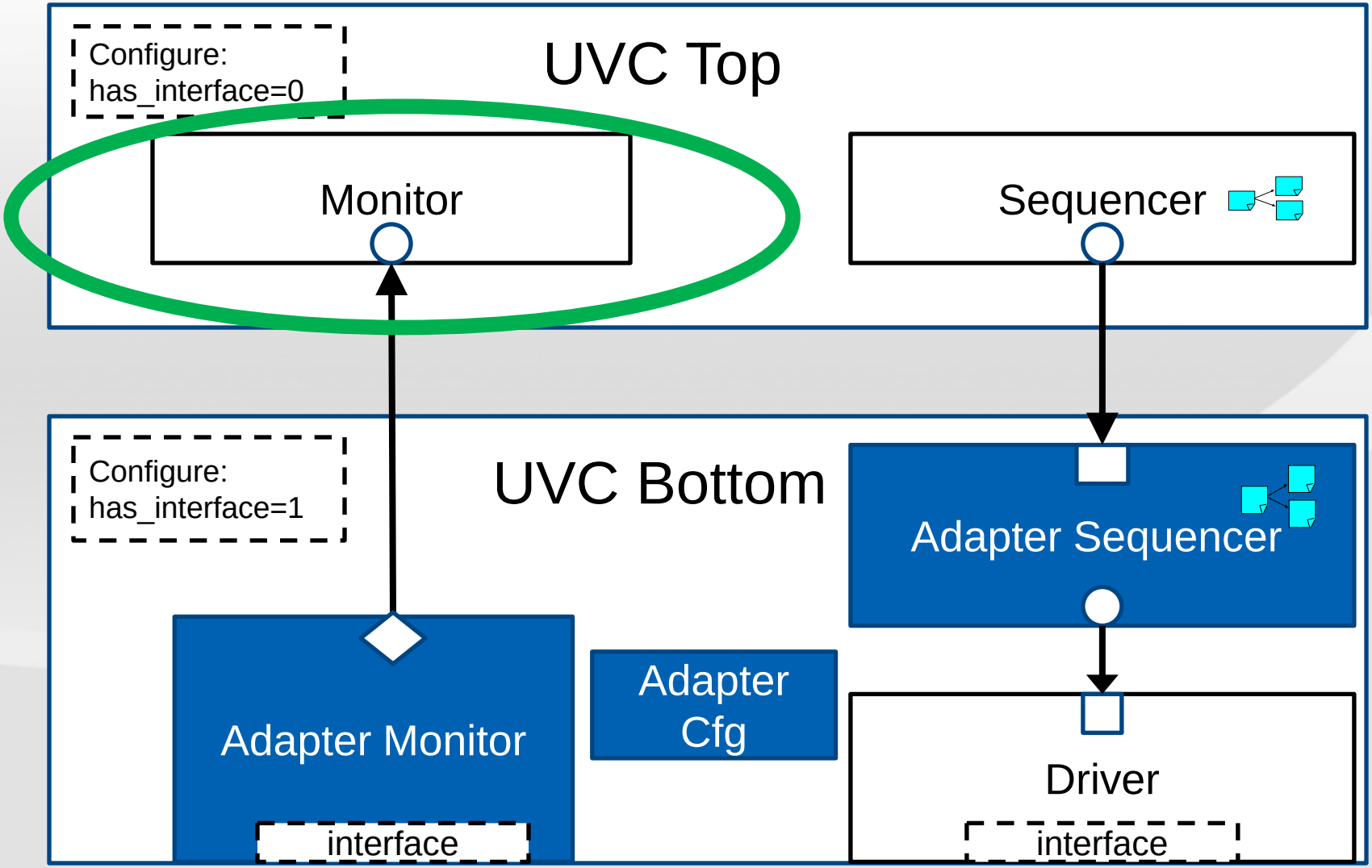
UVM

accellera SYSTEMS INITIATIVE

Implementation Option C



Monitor Code Snippet



Monitor Code (Top UVC)

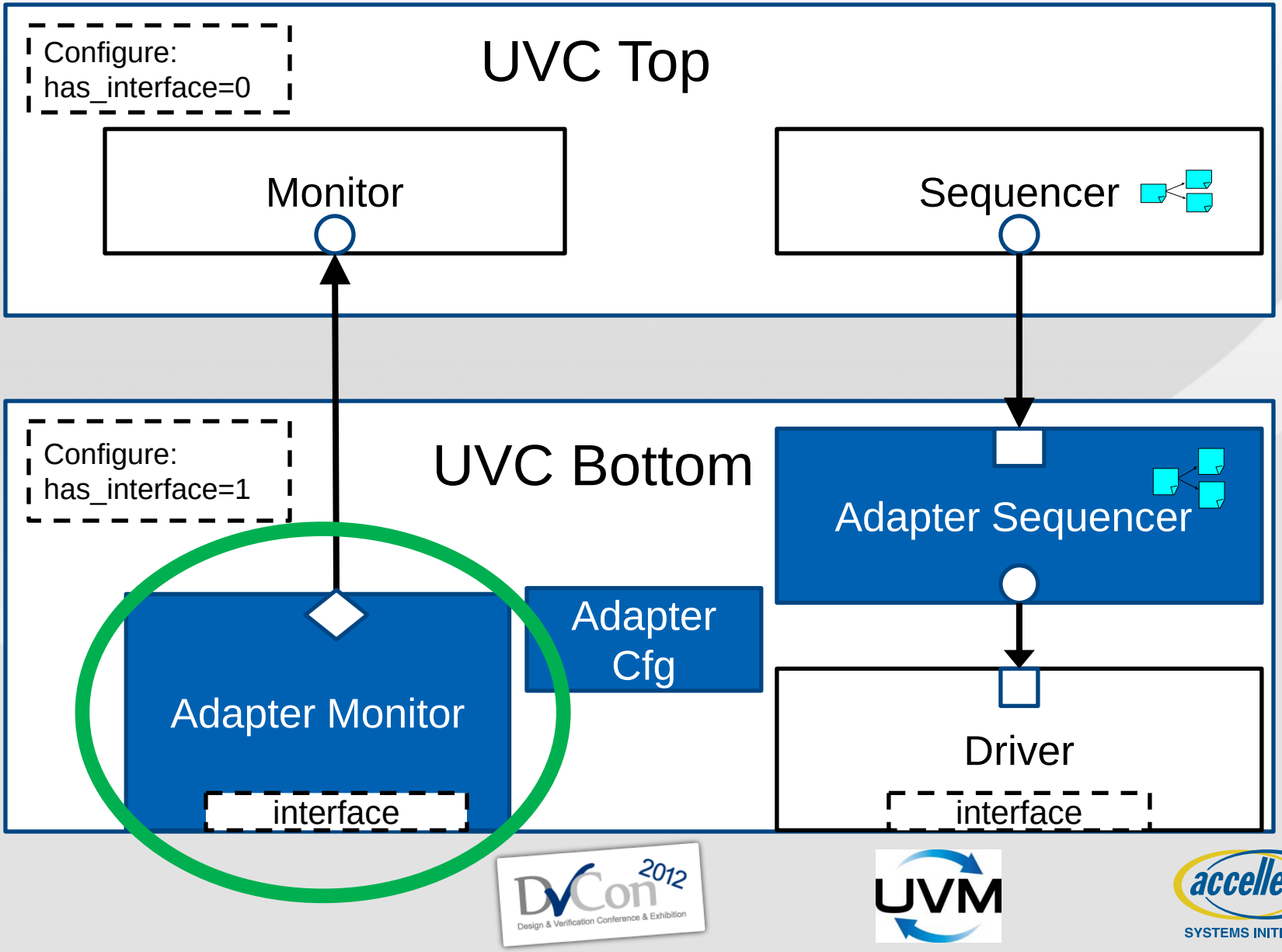
```
class top_monitor extends uvm_monitor;

uvm_analysis_imp#(top_trans, top_monitor) top_item_collected_imp;

function void write(top_trans trans);

  // Perform checks on the transaction & coverage
  …
  // Send this transaction to other components
  publish_transaction(trans);

endfunction : write

virtual function void publish_transaction(trans);
  $cast(cloned_trans, trans.clone());
  item_collected_port.write(cloned_trans);
endfunction : publish_transaction
```

Add TLM analysis imp port to allow lower layer UVC to send transactions to this UVC

Typical UVM Monitor Procedure code

Broadcast transactions to SB and other components

Monitor Adapter Code Snippet



Monitor Adapter Code (Bottom Adapter UVC extension)

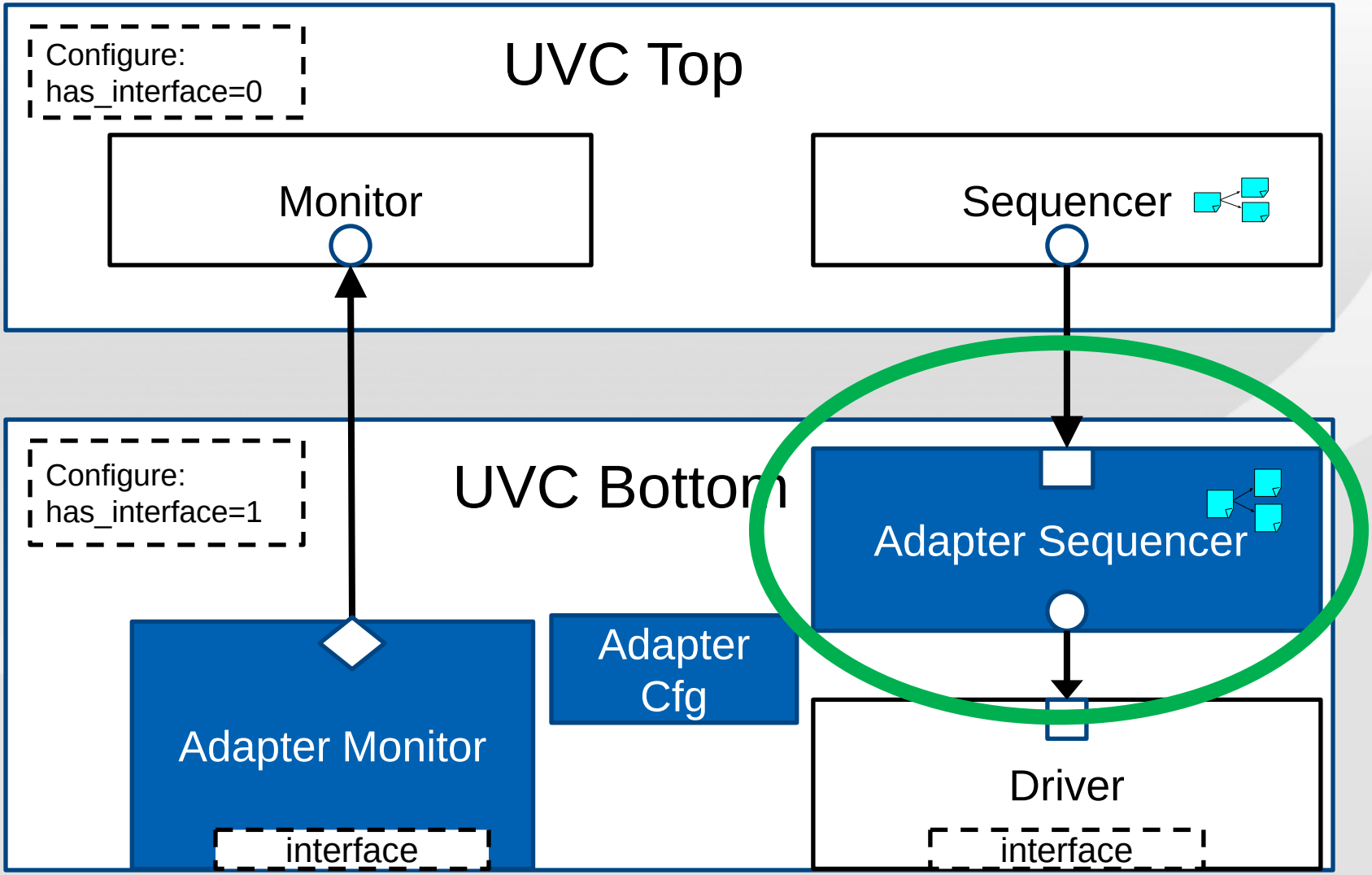
```
class btm_adapter_monitor extends btm_monitor;

uvm_analysis_port #(top_trans) top_item_collected_port;

virtual function void publish_transaction();
  super.publish_transaction();

  // top layer transactions
  top_trans = new;

  // USER: Add your convert code here
  …
  $cast(cloned_top_trans, top_trans.clone());
  top_item_collected_port.write(top_trans);
endfunction
…
```

Analysis Port that broadcasts **top transaction** to top UVC

Convert bottom transaction to top transaction and send it to top UVC

Send the top transaction to the top UVC

Sequencer Adapter Code Snippet



Sequencer Code (Bottom Adapter UVC extension)

```
class btm_adapter_sequencer extends btm_sequencer;

  //! This is the TLM pull port that allows this UVC to communicate with a top UVC
  uvm_seq_item_pull_port #(top_trans) top_seq_item_port;

  …
```

Use inheritance to extend the bottom UVC and add a top sequence item pull port

Sequence Code (Bottom Adapter UVC sequence)

```
class btm_adapter_sequence extends uvm_sequence #(btm_trans);

  btm_transaction btm_trans;

  top_transaction top_trans;

  `uvm_declare_p_sequencer(btm_adapter_sequencer)

task body();
 forever begin : repeat_block
   p_sequencer.top_seq_item_port.get_next_item(top_trans);
   // USER: Add your top/bottom conversion here
   `uvm_send(btm_trans)
   p_sequencer.top_seq_item_port.item_done();
  end
endtask
```

Declare bottom & top transaction

Forever convert top transaction to bottom transaction

Sequence Code (Bottom Adaptor With Response)

```
class btm_adapter_sequence extends uvm_sequence #(btm_trans);
  btm_transaction  btm_req_trans;
  top_transaction   top_req_trans;
  btm_transaction  btm_rsp_trans;
  top_transaction   top_rsp_trans;

  `uvm_declare_p_sequencer(btm_adapter_sequencer)

  task body();
    forever begin : repeat_block
      p_sequencer.top_seq_item_port.get_next_item(top_req_trans);
      // USER: Convert top request to bottom request here
      `uvm_send(btm_req_trans)
      get_response(btm_rsp_trans)
      top_rsp_trans = new;
      // USER: Convert bottom response to top response here
      p_sequencer.top_seq_item_port.item_done(top_rsp_trans);
    end
  endtask
```

Declare bottom & top

Declare bottom & top response transactions

Same forever loop

Get bottom response, convert to top

Pass top response to item_done()

Testbench Override Code

```
class testbench extends uvm_env;

function void build_phase(uvm_phase phase);

  set_type_override_by_type (
                        btm_cfg::get_type(),
                        btm_adapter_cfg::get_type());
  set_inst_override_by_type("btm.*",
                        btm_sequencer::get_type(),
                        btm_adapter_sequencer::get_type());
  set_inst_override_by_type("btm.*",
                        btm_monitor::get_type(),
                        btm_adapter_monitor::get_type());
endfunction
…
```

Use factory to override UVC bottom components with “bottom adapter” components.

Testbench Override Code (Cont)

```
connect_phase(uvm_phase phase);
  btm_adapter_monitor monitor;
  btm_adapter_sequencer sequencer;

  $cast(monitor, btm.monitor);
  monitor.top_item_collected_port.connect(top.monitor.top_item_collected_imp);

  if (top.is_active == UVM_ACTIVE && btm.is_active == UVM_ACTIVE) begin
    $cast(sequencer, btm.sequencer);
    sequencer.top_seq_item_port.connect(top.sequencer.seq_item_export);
  end
...
```

Connect the monitor adapter TLM ports

Connect the sequencer adapter TLM ports

Testbench Override Code (Cont)

```
connect_phase(uvm_phase phase);
  …
  uvm_config_db#(uvm_object_wrapper)::set(this,
                                          (".run_phase",
                                           "default_sequence",
                                            btm_adapter_sequence::type_id::get());
  …
```

Start the adapter sequence – it will run continuously during all the UVM runtime phases

MITRE Results

- **14 unit and one system testbench**
  - 8 engineers over 6 months
- **Heavy stimulus reuse with abstract UVCs**
  - Easier to understand, create, and share
  - 2, 3 and 4 stacked UVCs
- **Full bottom UVC reuse between unit level**
  - Every unit had a different logical use
- **Reused unit environments entirely in system**
  - Modified connections for system topology

Conclusion

- **UVM provides no UVC (Agent) layering methodology**
- **Stacking UVCs == reusable UVCs and stimulus**
  - Simple API and implementation
- **Proven results with this and successive projects at MITRE and beyond**
- **Full example contributed on uvmworld.org**