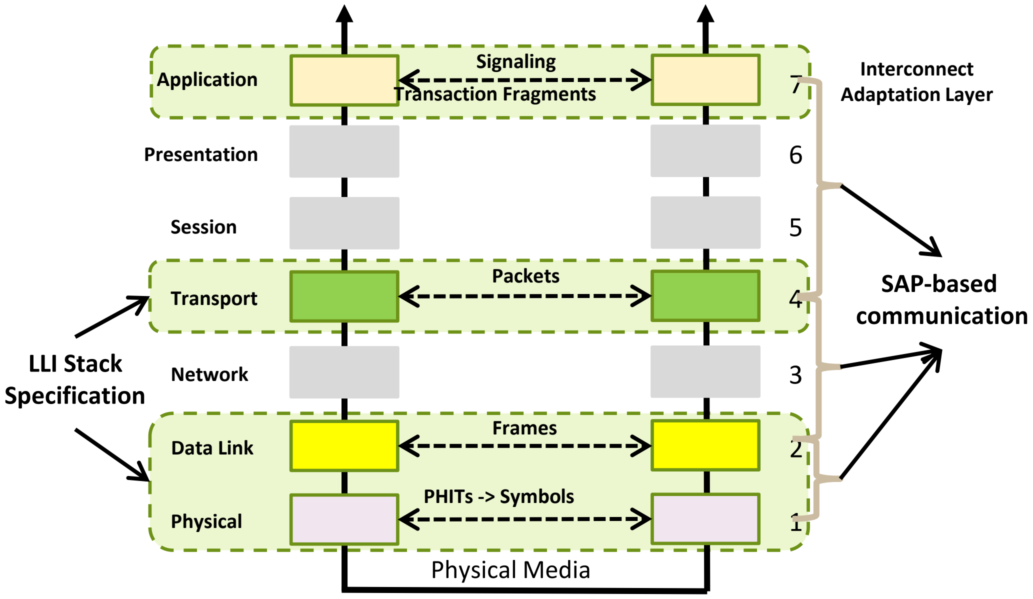
Verifying Layered Protocols – Leveraging Advanced UVM Capabilities

Parag Goel
Sr. Corporate Application Engineer
Synopsys India Pvt. Ltd.

Stack based architecture

– Reasons for adoption

- Breaking network designs into functional layers
  - Enables each layer to be governed by simple protocols
  - Each with a few well-defined tasks



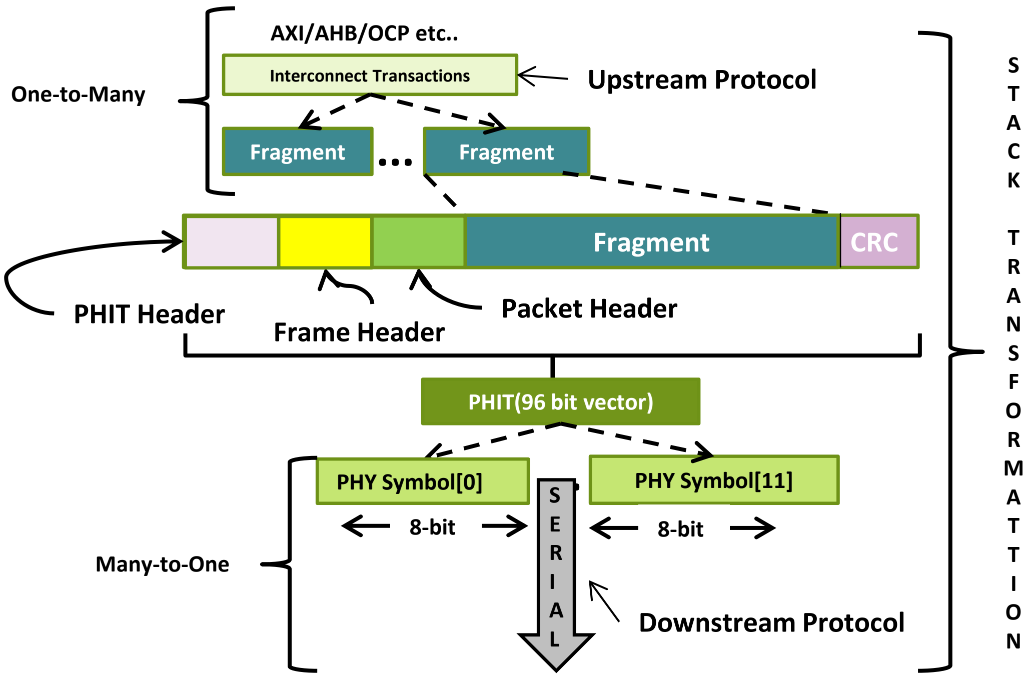
- Easier Debug
  - Faster convergence to where network failures originate
- The same user-level (application) program can be used over diverse communication networks.
  - Same WWW browser can be used when you are connected to the internet via a LAN or a dial-up line.

Agenda

- Verification IP & Testbench Challenges
  - Architectural Challenges
    - Stimulus generation - Generate varied traffic corresponding to each layer
    - Visibility and granularity of control - retrieving transaction for analysis
    - Support for intermediate layer and multi-lane scenarios
  - Application Challenges
    - Verifying Transformations
    - Graceful End-of-test
    - Enough debugging hooks
- Need to map verification challenges to existing methodologies
  - Leverage available methodology capabilities
  - Build intelligent layers around base classes for more powerful verification setup

Architectural Challenges - I

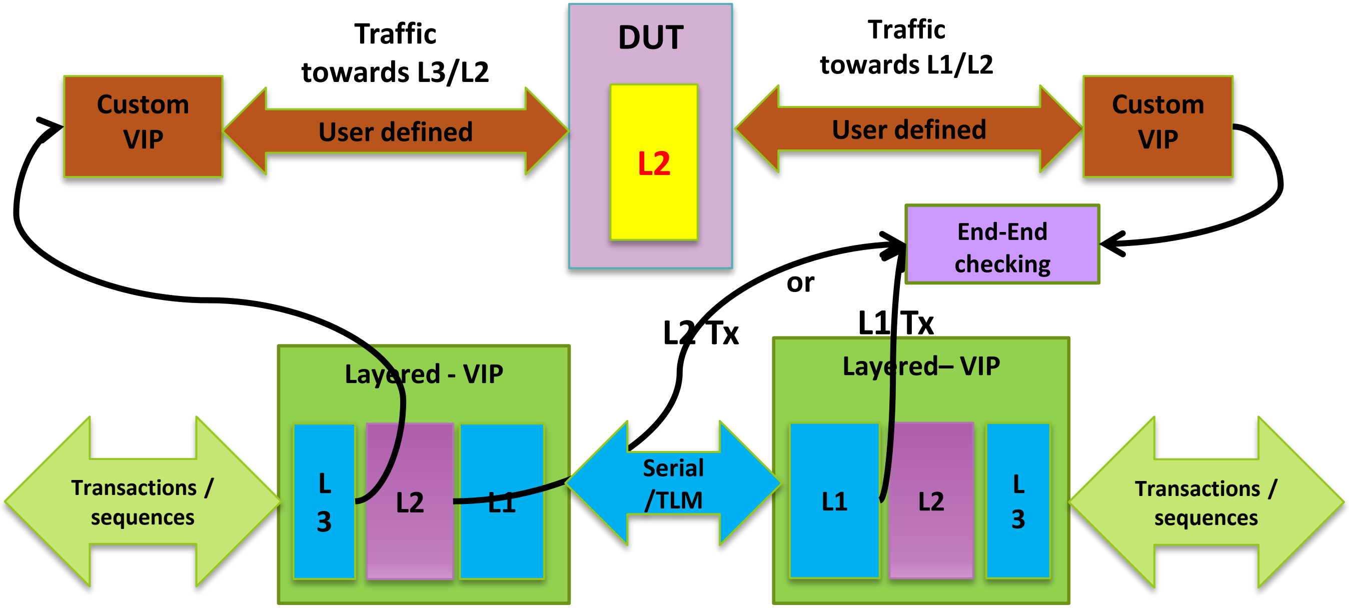
- Stimulus Generation



- Ability to inject stimulus at any layer
  - It should be possible to configure any of the layers as top-most layer generating the highest upstream sequence.
- Ability to arbitrate, i.e. mix and match stimulus from the upper layer as well as from the testbench
- Ability to retrieve and modify the transaction at any specific layer via callbacks, factory mechanism and UVM command-line override.

Architectural Challenges - II

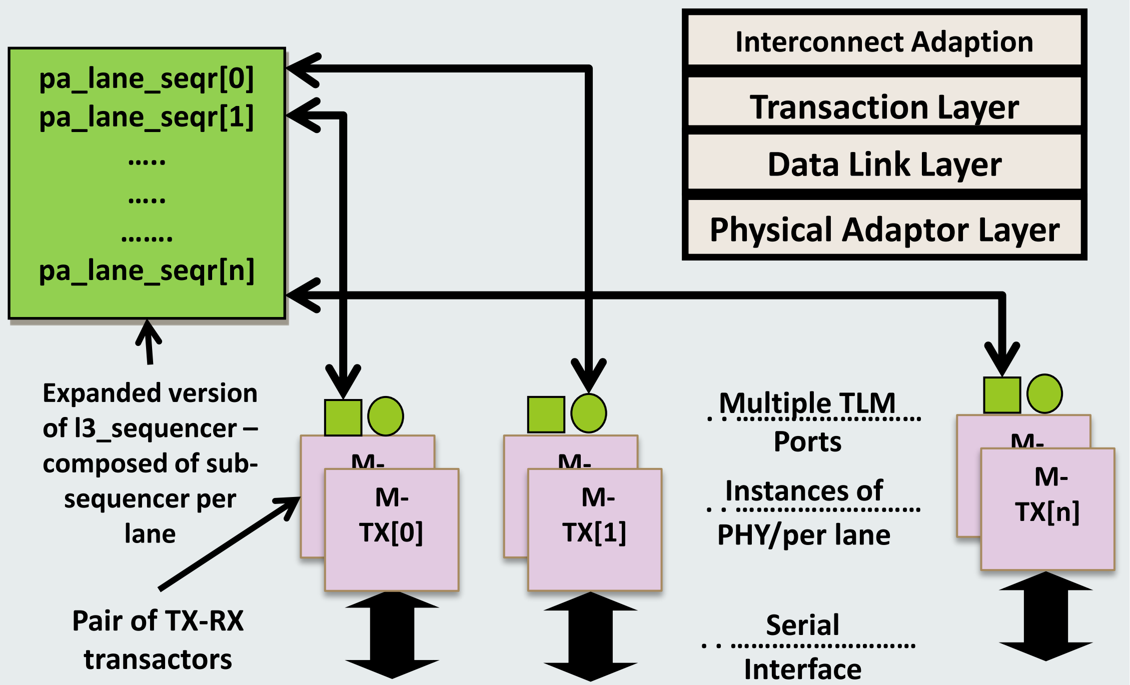
- Verifying intermediate layer



- Appropriate hooks/callbacks/ports for each component
  - Retrieve the transactions from any layer
- Provision to hook up intermediate custom drivers which can then drive the interface between the layers.
- Callbacks across monitors to verify transformations across the layers.

Architectural Challenges - III

- *Verifying multi-lane scenarios*



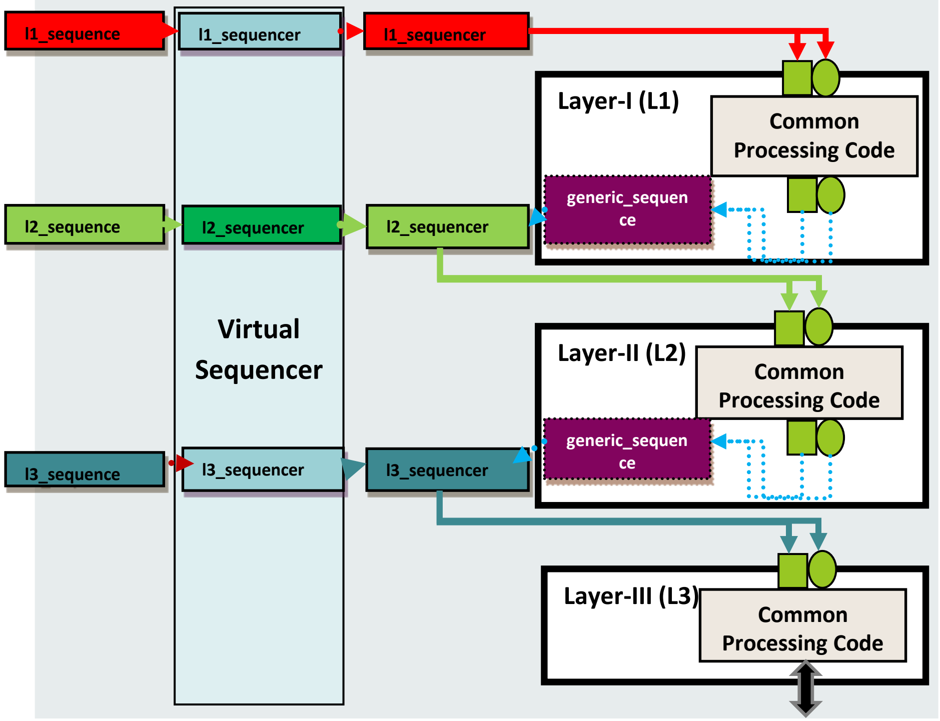
```
→ Sequencer Definition←
typedef class
   uvm_sequencer#(mphy_transfer)
                 mphy_sequencer;
```

```
class lli_agent extends uvm_agent;
 → Sequencer Handle Declaration – Dynamic Array ←
  mphy_sequencer pa_lane_seqr[];

  function void build_phase(uvm_phase phase);
    if(!uvm_config_db#(lli_configuration)::get(this, “”, “cfg”, cfg)) `uvm_fatal(………….)
   → Sequencer creation ←
    pa_lane_seqr = new[cfg.tx_lane_count];
    foreach(pa_lane_seqr[i])
      pa_lane_seqr[i] = mphy_sequencer::type_id::create($sformatf(“pa_lane_seqr[%0d]”, i), this);
  endfunction
endclass
```

Addressing Architectural challenges - I

- Generic architecture



→ Class & UVM factory Registration ←

```
class generic_sequence#(type REQ=uvm_seqence_item,
type RSP=REQ) extends uvm_sequence#(REQ,RSP);

`uvm_object_param_utils(generic_sequence#(REQ,RSP))
```

→ Parent Sequencer Declaration ←

```
 `uvm_declare_p_sequencer(uvm_sequencer#(REQ))
```

→ Transaction Handle ←

```
local REQ req;
```

→ Dispatch: Drive REQ on downstream sequencer ←

```
 task dispatch(uvm_seqeuncer#(REQ) seqr, REQ req);
   this.req = req;
   this.start(seqr);
 endtask
```

→ Body method – Initiate REQ ←

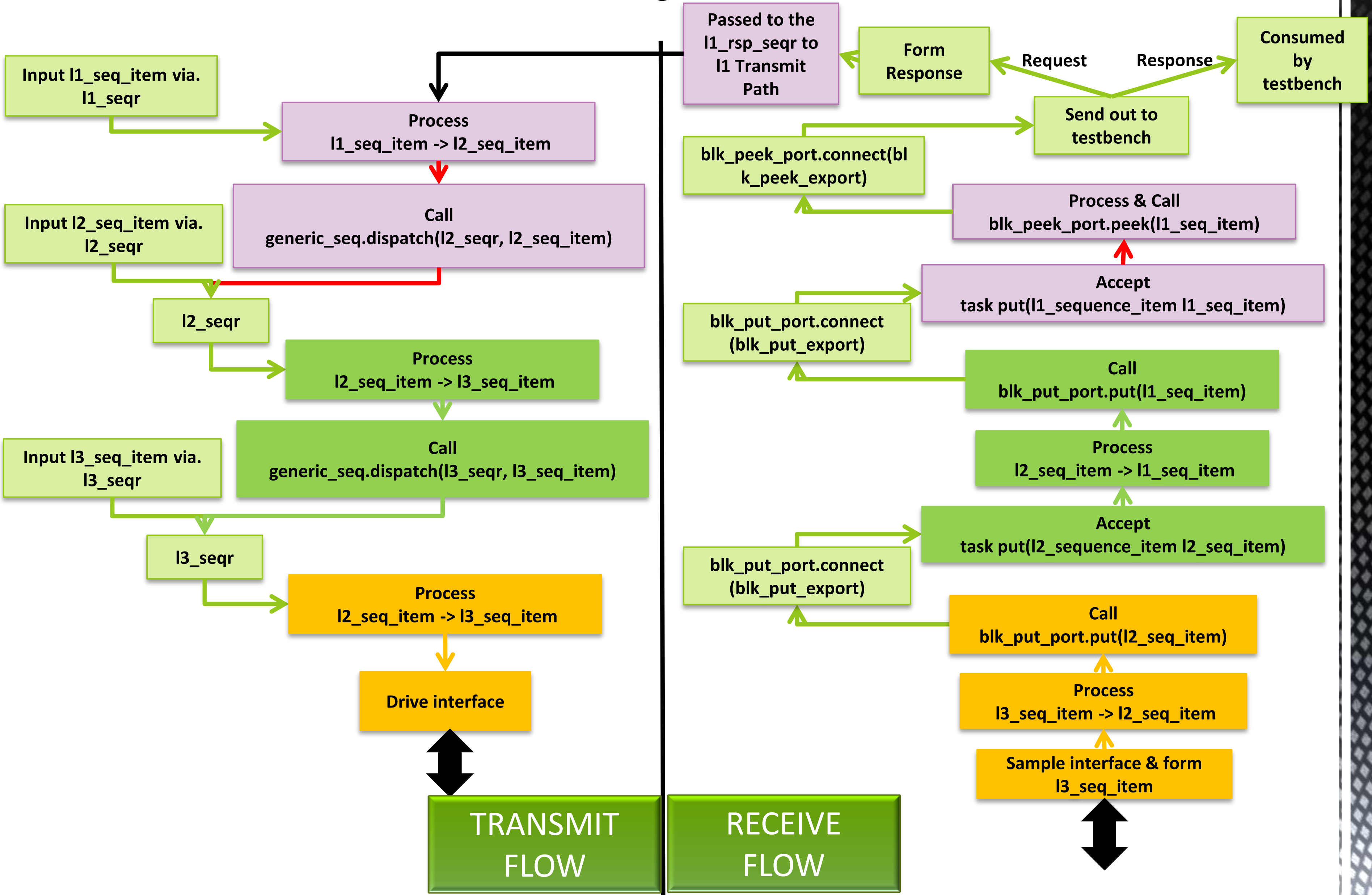
```
task body();
  if (this.req != null) begin
    this.wait_for_grant();
    this.send_request(req);
    this.wait_for_item_done();
  end
endtask
```

→ response_handler method – To discard RSP ←

```
function void response_handler
                 (uvm_sequence_item response);
/* Just drop the response. */
endfunction
endclass
```

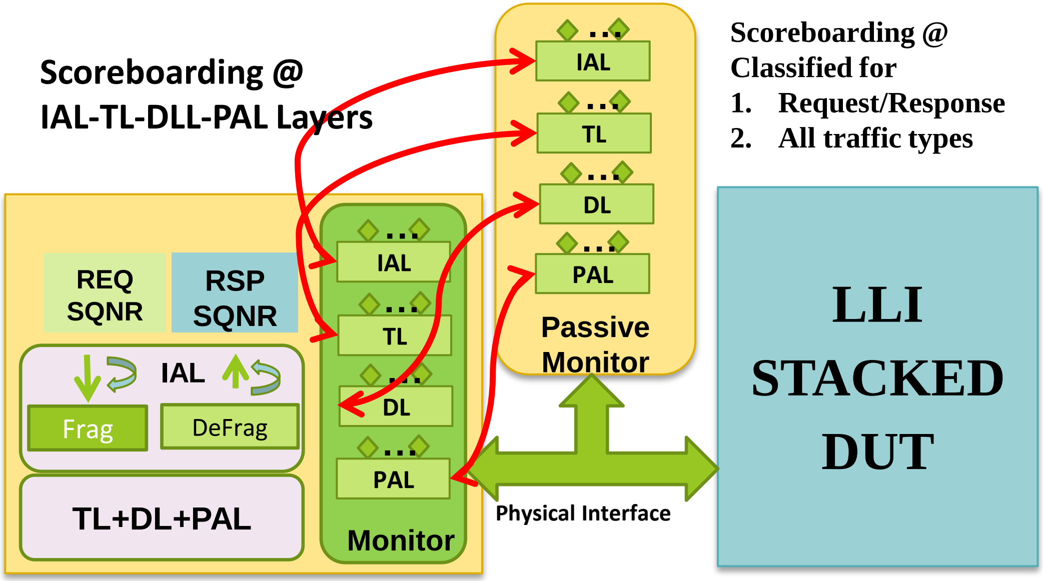
Addressing Architectural challenges - II

- Generic architecture – Flow Diagram


Input l1_seq_item via. l1_seqr
Process l1_seq_item -> l2_seq_item
Input l2_seq_item via. l2_seqr
Call generic_seq.dispatch(l2_seqr, l2_seq_item)
l2_seqr
Process l2_seq_item -> l3_seq_item
Input l3_seq_item via. l3_seqr
Call generic_seq.dispatch(l3_seqr, l3_seq_item)
l3_seqr
Process l2_seq_item -> l3_seq_item
Drive interface
TRANSMIT FLOW
RECEIVE FLOW
Passed to the l1_rsp_seqr to l1 Transmit Path
Form Response
Request
Response
Consumed by testbench
Send out to testbench
blk_peek_port.connect(blk_peek_export)
Process & Call blk_peek_port.peek(l1_seq_item)
Accept task put(l1_sequence_item l1_seq_item)
blk_put_port.connect (blk_put_export)
Call blk_put_port.put(l1_seq_item)
Process l2_seq_item -> l1_seq_item
Accept task put(l2_sequence_item l2_seq_item)
blk_put_port.connect (blk_put_export)
Call blk_put_port.put(l2_seq_item)
Process l3_seq_item -> l2_seq_item
Sample interface & form l3_seq_item

Application Challenges - I

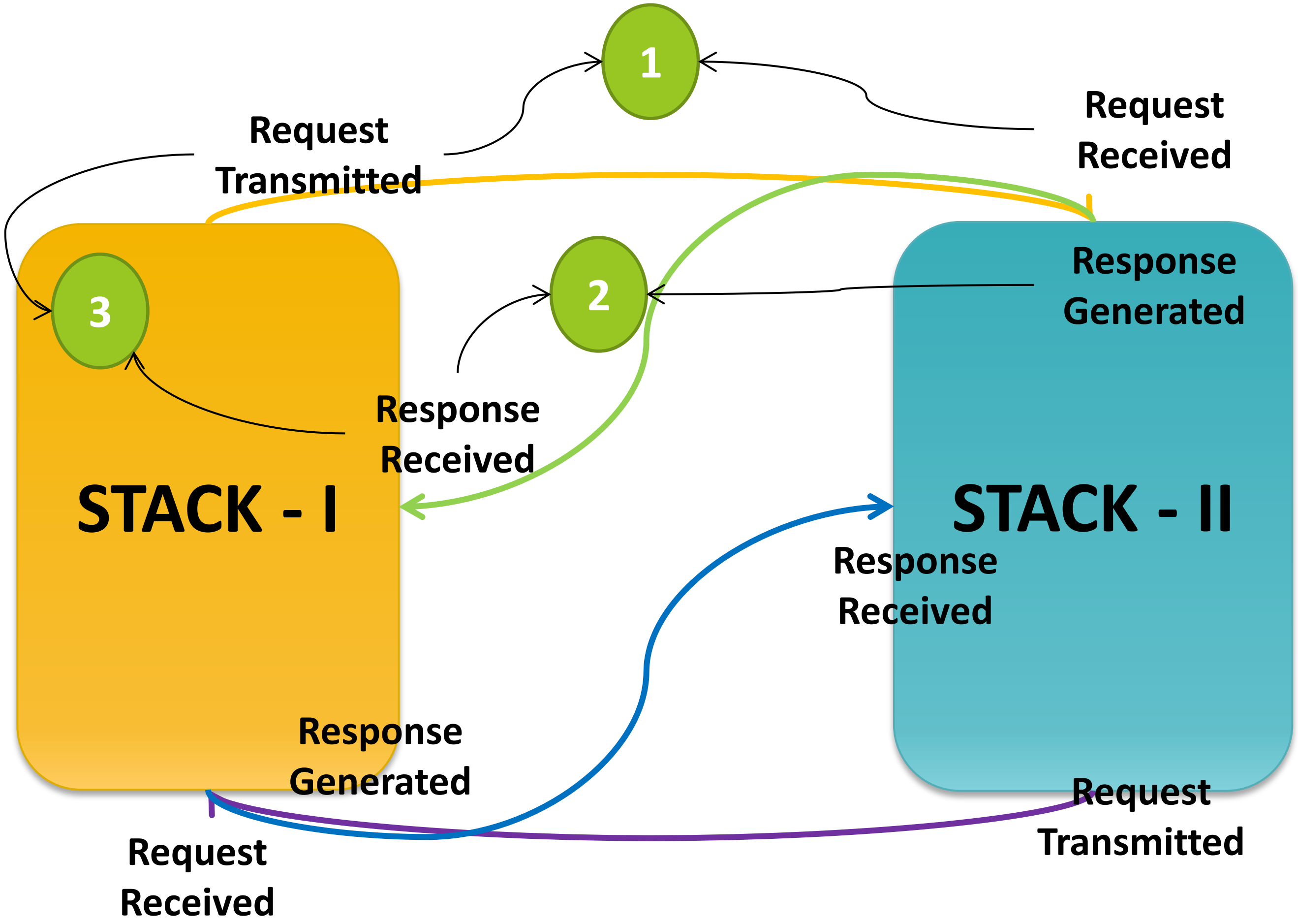
- Scoreboarding Challenges - I



- Transformations that would need to be verified
  - End to end transformations in the ‘transmit’ and ‘receive’ paths
  - Transformations across all the traffic types
  - For requests / responses

Application Challenges - I

- Scoreboarding Challenges - Checking for equivalence



Addressing Scoreboarding Challenges

Policy Based design :

- template taking several type parameters
- specialized to encapsulates orthogonal aspects of the behavior of the instantiated class

```
class lli_comp #(type T = int);
 static uvm_comparer relevant_comparer = new();
 static function bit comp(input T a, input T b);
  relevant_comparer.physical = 1;
  relevant_comparer.abstract = 0;
  return a.compare(b, relevant_comparer);
 endfunction
endclass
```

```
 class lli_scoreboard#(type T=uvm_sequence_item)
                                extends uvm_scoreboard;

 → Export Transmitted/Received Requests ←
 uvm_analysis_export#(T) tx_export, rx_export;
 → "in order comparator" ←
 uvm_in_order_comparator #(T, lli_comp#(T),
         uvm_class_converter#(T)) comparator;

 → Building components ←
 → Connect local export to comparator export ←
 → Reporting results ←
```

```
class lli_system_env extends uvm_env;

typedef lli_scoreboard#(svt_mipi_lli_transaction)
trans_scbd;
typedef lli_scoreboard#(svt_mipi_lli_packet)
pkt_scbd;

→ An instance of VIP AGENT - LLI Master/Slave ←
svt_mipi_lli_agent mstr, slv;

→ IAL Scoreboard Instances for request xact ←
trans_scbd m_s_ll_xact_sb;
trans_scbd m_s_be_xact_sb;

→ Construct the IAL scoreboard instances ←
 m_s_ll_xact_sb = new("m_s_ll_xact_sb", this);
 m_s_be_xact_sb = new("m_s_be_xact_sb", this);

→ Connect the monitor to the scoreboard for Master
LL request ←
mstr.ial_mon.tx_ll_ta_xact_observed_port.
                 connect(m_s_ll_xact_sb.tx_export);
slv.ial_mon.rx_ll_in_xact_observed_port.
                 connect(m_s_ll_xact_sb.rx_export);

→ Connect the monitor to the scoreboard for Master
BE request ←
mstr.ial_mon.tx_be_ta_xact_observed_port.
                 connect(m_s_be_xact_sb.tx_export);
slv.ial_mon.rx_be_in_xact_observed_port.
                 connect(m_s_be_xact_sb.rx_export);
endclass
```

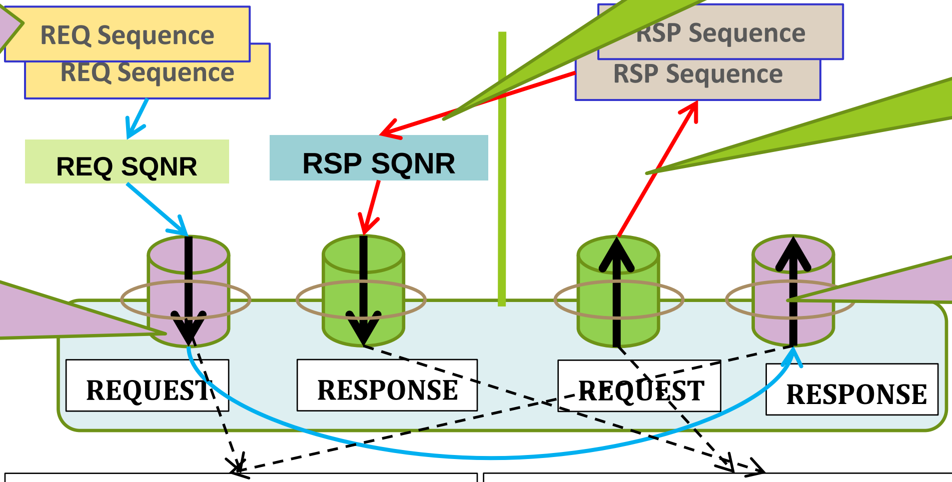
Application Challenges - II

- End-of-test Mechanism

For a reactive sequence, drop the objection - response passed down on the transmit path

Raise/drop objection in the *req_sequence's pre_body()* and *post_body()*

Raise an objection when request appears on the receive path.



Raise an objection in a callback - the request gets accepted by the highest layer

Drop objections once the peer stack generates the response and the same is received

TRANSMIT PATH:
Raise an objection when new REQ started
Drop an objection when RSP is received

RECEIVE PATH:
Raise an objection when new REQ received
Drop an objection when RSP is transmitted

Drain Time - Amount of time to wait once all objections have been dropped

```
int stack_round_trip_time;

task main_phase(uvm_phase phase);
  phase.phase_done.set_drain_time
     (this, 2*stack_round_trip_time);
endtask
```

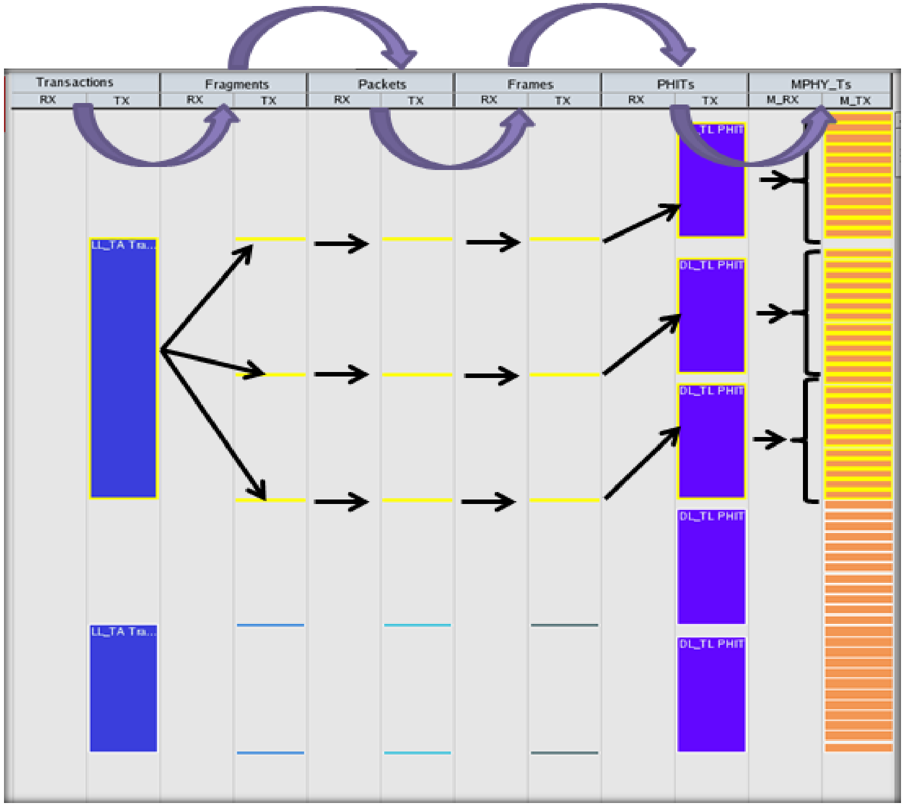
Global timeout - The phase ends if the timeout expires even before all objections are dropped

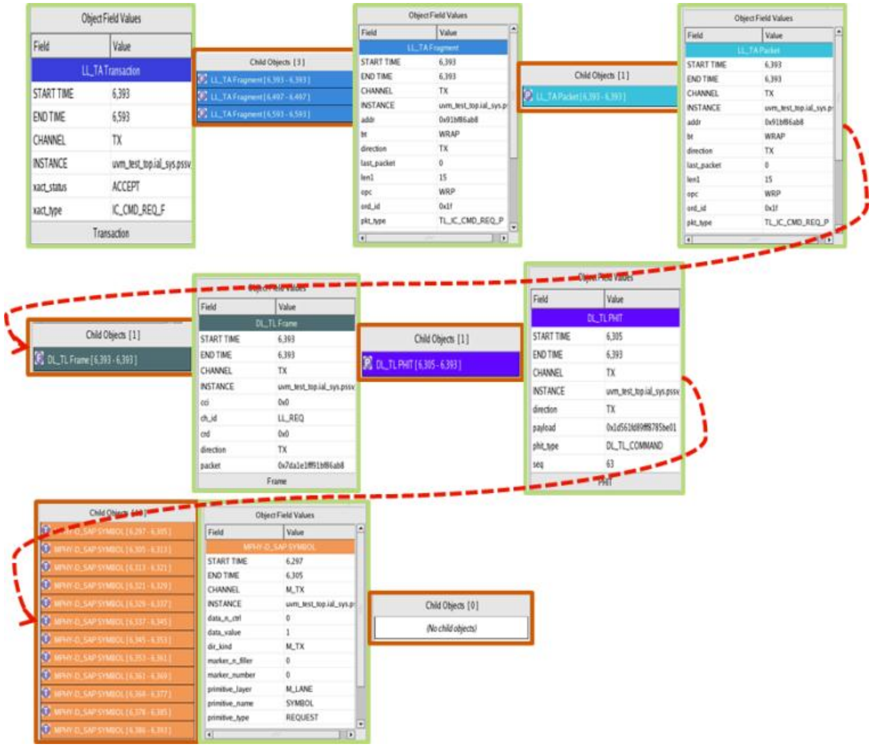
```
`define HS_MODE_GLOBAL_TIMEOUT 5ms
`define LS_MODE_GLOBAL_TIMEOUT 25ms

function void test_base::build_phase(...);
  ➔ Set the global timeout ⬅
  if(sys_cfg.mode == LS_MODE)
    set_global_timeout(`LS_MODE_GLOBAL_TIMEOUT);
  else
    set_global_timeout(`HS_MODE_GLOBAL_TIMEOUT);
endfunction
```

Application Challenges - III

- Debug Abstraction





Object Field Values

Field	Value
LL_TA Transaction	
START TIME	6,393
END TIME	6,593
CHANNEL	TX
INSTANCE	uvm_test_top.ial_sys.pssv
xact_status	ACCEPT
xact_type	IC_CMD_REQ_F

Transaction

Child Objects [3]

- LL_TA Fragment [6,393 - 6,393]
- LL_TA Fragment [6,497 - 6,497]
- LL_TA Fragment [6,593 - 6,593]

Object Field Values

Field	Value
LL_TA Fragment	
START TIME	6,393
END TIME	6,393
CHANNEL	TX
INSTANCE	uvm_test_top.ial_sys.p
addr	0x91bf86ab8
bt	WRAP
direction	TX
last_packet	0
len1	15
opc	WRP
ord_id	0x1f
pkt_type	TL_IC_CMD_REQ_P

Child Objects [1]

- LL_TA Packet [6,393 - 6,393]

Object Field Values

Field	Value
LL_TA Packet	
START TIME	6,393
END TIME	6,393
CHANNEL	TX
INSTANCE	uvm_test_top.ial_sys.p
addr	0x91bf86ab8
bt	WRAP
direction	TX
last_packet	0
len1	15
opc	WRP
ord_id	0x1f
pkt_type	TL_IC_CMD_REQ_P

Child Objects [1]

- DL_TL Frame [6,393 - 6,393]

Object Field Values

Field	Value
DL_TL Frame	
START TIME	6,393
END TIME	6,393
CHANNEL	TX
INSTANCE	uvm_test_top.ial_sys.pssv
cci	0x0
ch_id	LL_REQ
crd	0x0
direction	TX
packet	0x7da1e1ff91bf86ab8

Frame

Child Objects [1]

- DL_TL PHIT [6,305 - 6,393]

Object Field Values

Field	Value
DL_TL PHIT	
START TIME	6,305
END TIME	6,393
CHANNEL	TX
INSTANCE	uvm_test_top.ial_sys.pssv
direction	TX
payload	0x1d561fd89ff8785be01
phit_type	DL_TL_COMMAND
seq	63

PHIT

Child Objects [[illegible]]

- MPHY-D_SAP SYMBOL [6,297 - 6,305]
- MPHY-D_SAP SYMBOL [6,305 - 6,313]
- MPHY-D_SAP SYMBOL [6,313 - 6,321]
- MPHY-D_SAP SYMBOL [6,321 - 6,329]
- MPHY-D_SAP SYMBOL [6,329 - 6,337]
- MPHY-D_SAP SYMBOL [6,337 - 6,345]
- MPHY-D_SAP SYMBOL [6,345 - 6,353]
- MPHY-D_SAP SYMBOL [6,353 - 6,361]
- MPHY-D_SAP SYMBOL [6,361 - 6,369]
- MPHY-D_SAP SYMBOL [6,369 - 6,377]
- MPHY-D_SAP SYMBOL [6,377 - 6,385]
- MPHY-D_SAP SYMBOL [6,385 - 6,393]

Object Field Values

Field	Value
MPHY-D_SAP SYMBOL	
START TIME	6,297
END TIME	6,305
CHANNEL	M_TX
INSTANCE	uvm_test_top.ial_sys.p
data_n_ctrl	0
data_value	1
dir_kind	M_TX
marker_n_filler	0
marker_number	0
primitive_layer	M_LANE
primitive_name	SYMBOL
primitive_type	REQUEST

Child Objects [0]

(No child objects)

- Tracing the transformation across each layer
  - Needs to be captured through TLM ports and dumped for Post processing
- Debug abstraction : Dumping of protocol objects
- Use transaction IDs to map across transformations

Summary

- Layered architecture in network protocols bring in advanced functionalities but complex verification challenges
  - Can be mapped across multiple new protocols (the MIPI family, PCIe, USB etc) and network designs
- UVM base classes provides the infrastructure on which required capabilities can be built
  - user defined enhancing sequence layering, phase completion tracking, transformation monitoring
- Verification infrastructure should continue to evolve with added complexity in design