2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Practical Schemes to Enhance Vertical, Horizontal and Platform Reusability of Verification Components in AMBA Based SoC Design

Ieryung Park(ieryung.park@sk.com)

Nara Cho

Yonghee Im

SK hynix

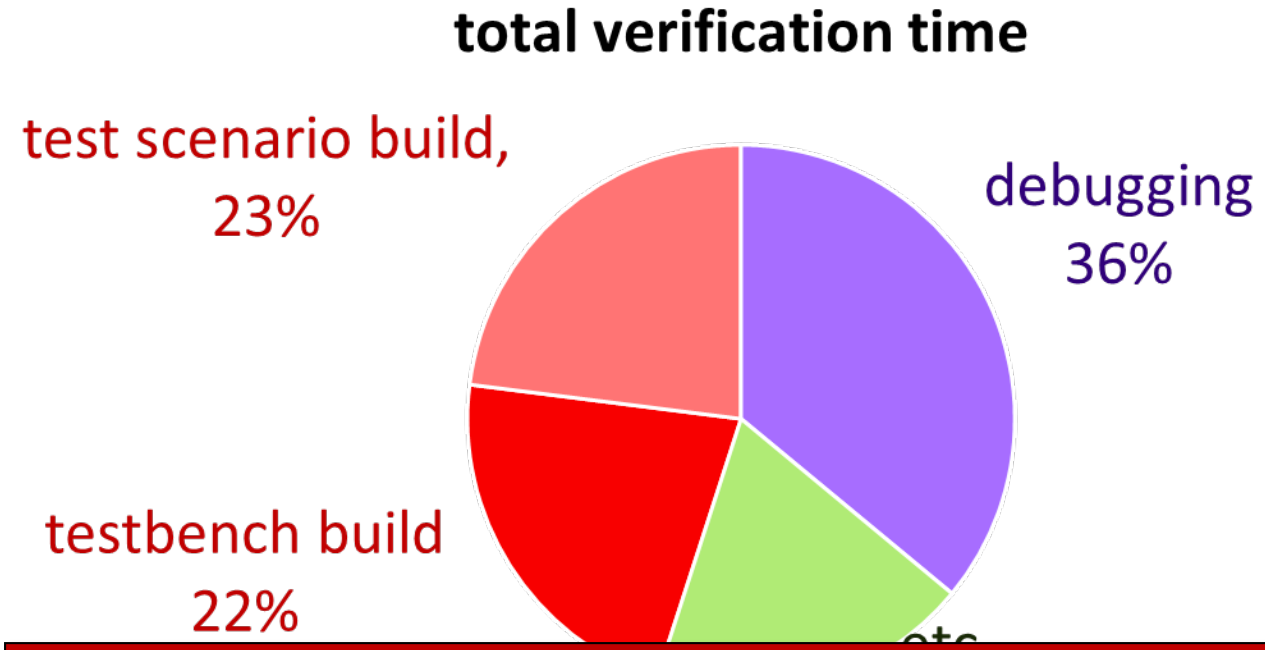
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- Reuse Problems
  - Testbench Build
  - Platform Reusability
  - Vertical & Horizontal Reusability
- AMBA Unified Verification System
- Firmware-Like Sequence with Hardware Abstraction Layer
- Conclusions
- Q&A

Introduction

- Verification Time



45% of Total Verification Time Is Spent on Preparing Test Environment †

† Hans van der Schoot, "UVM & Emulation", Mentor, 2014.

System Verilog & UVM

- Reusability of Verification Components

Object Oriented Programming Language

standardized verification methodology
reusable library

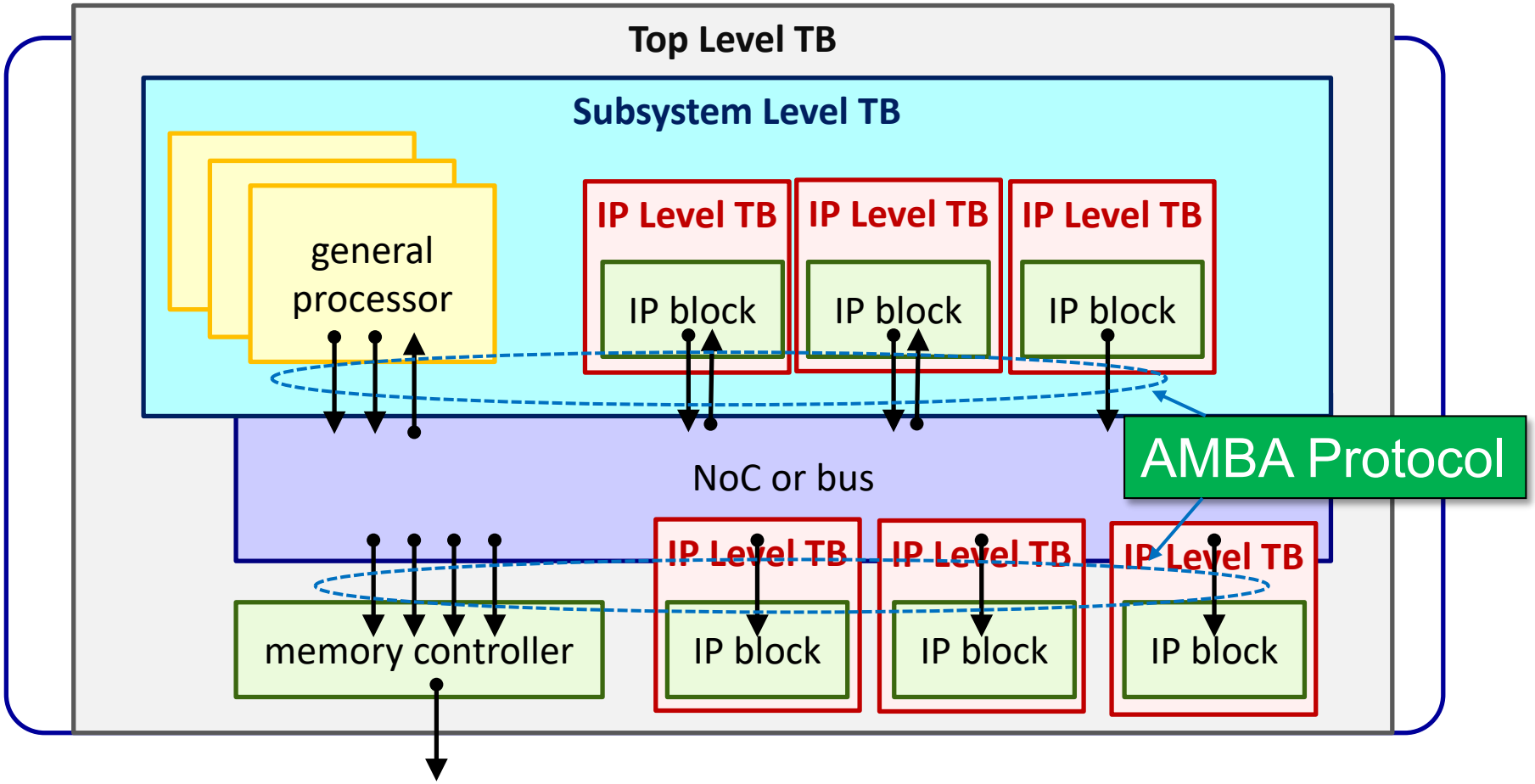
verification components

based on

CANNOT Ensure to REUSE

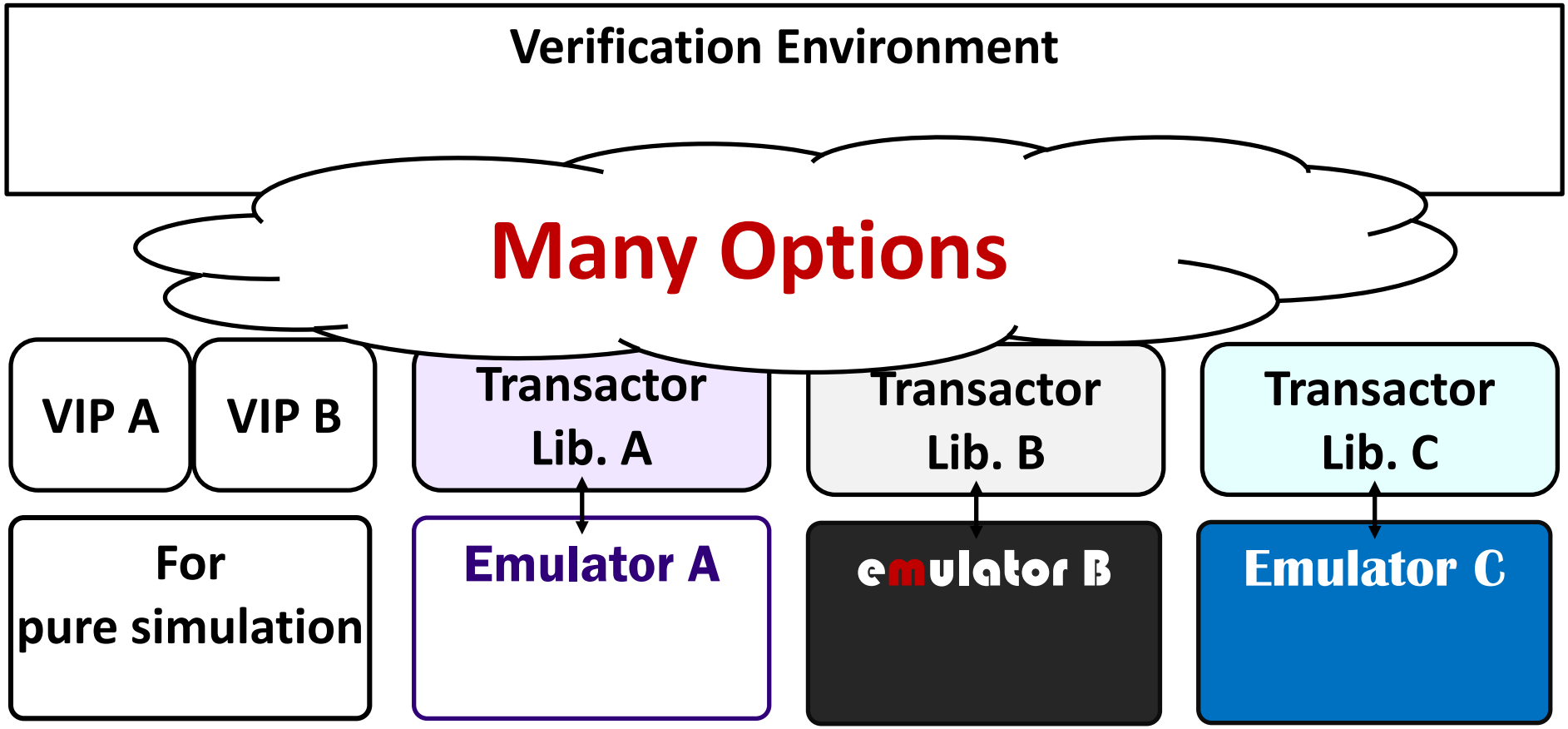
AMBA Based SoC

- General Form of SoC



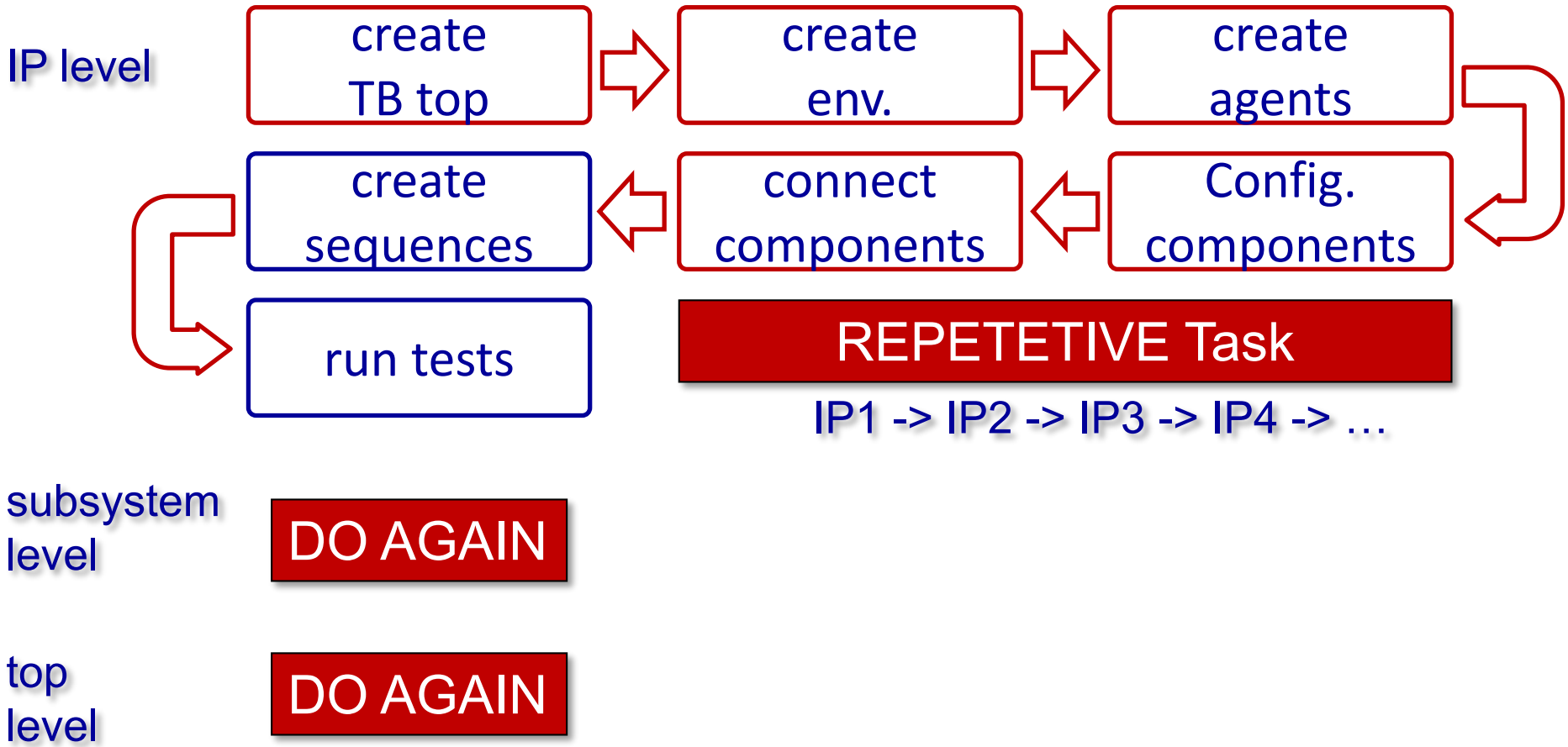
Various Libraries

- Libraries We Might Use



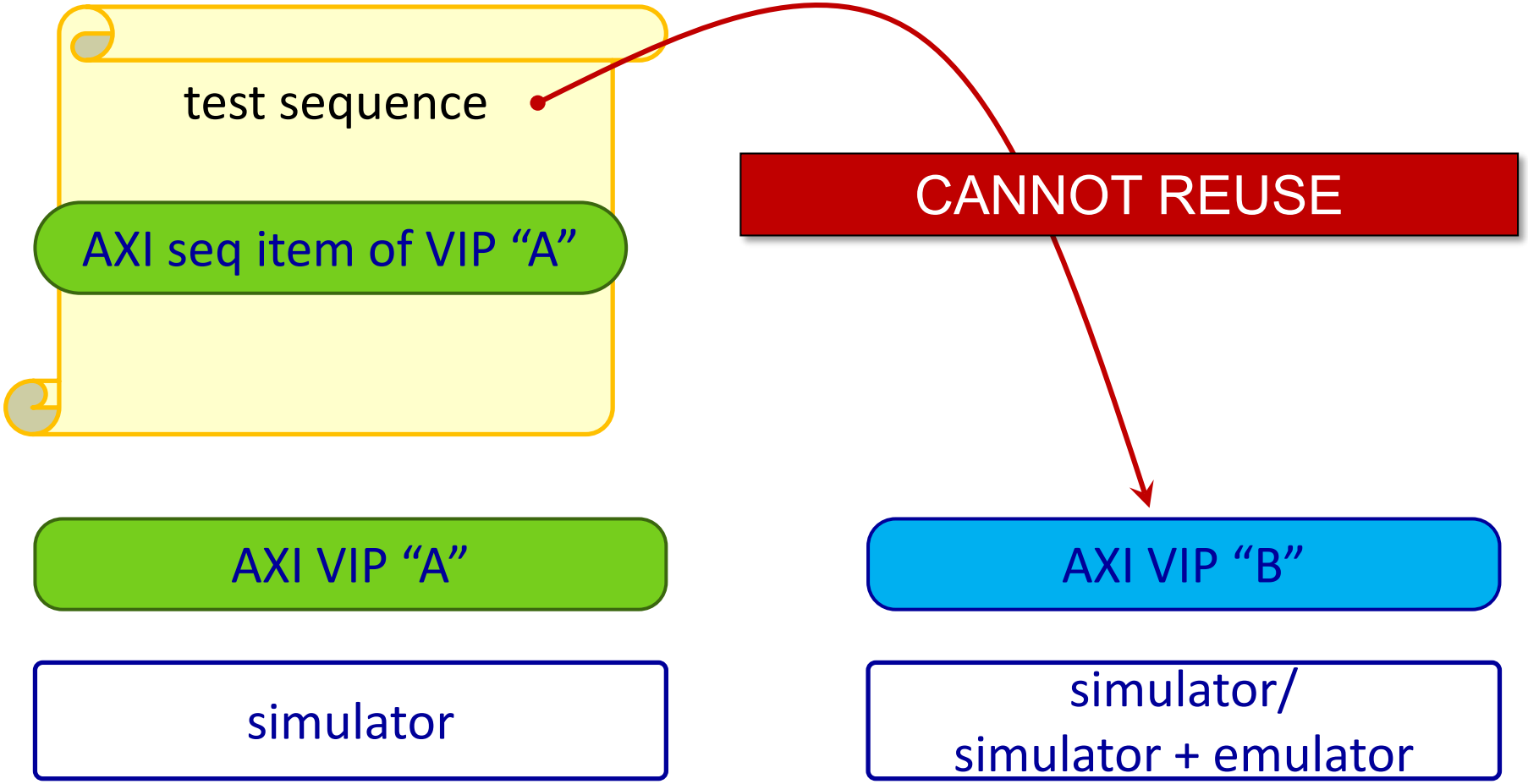
Reuse Problems : Testbench Build

- Verification Process



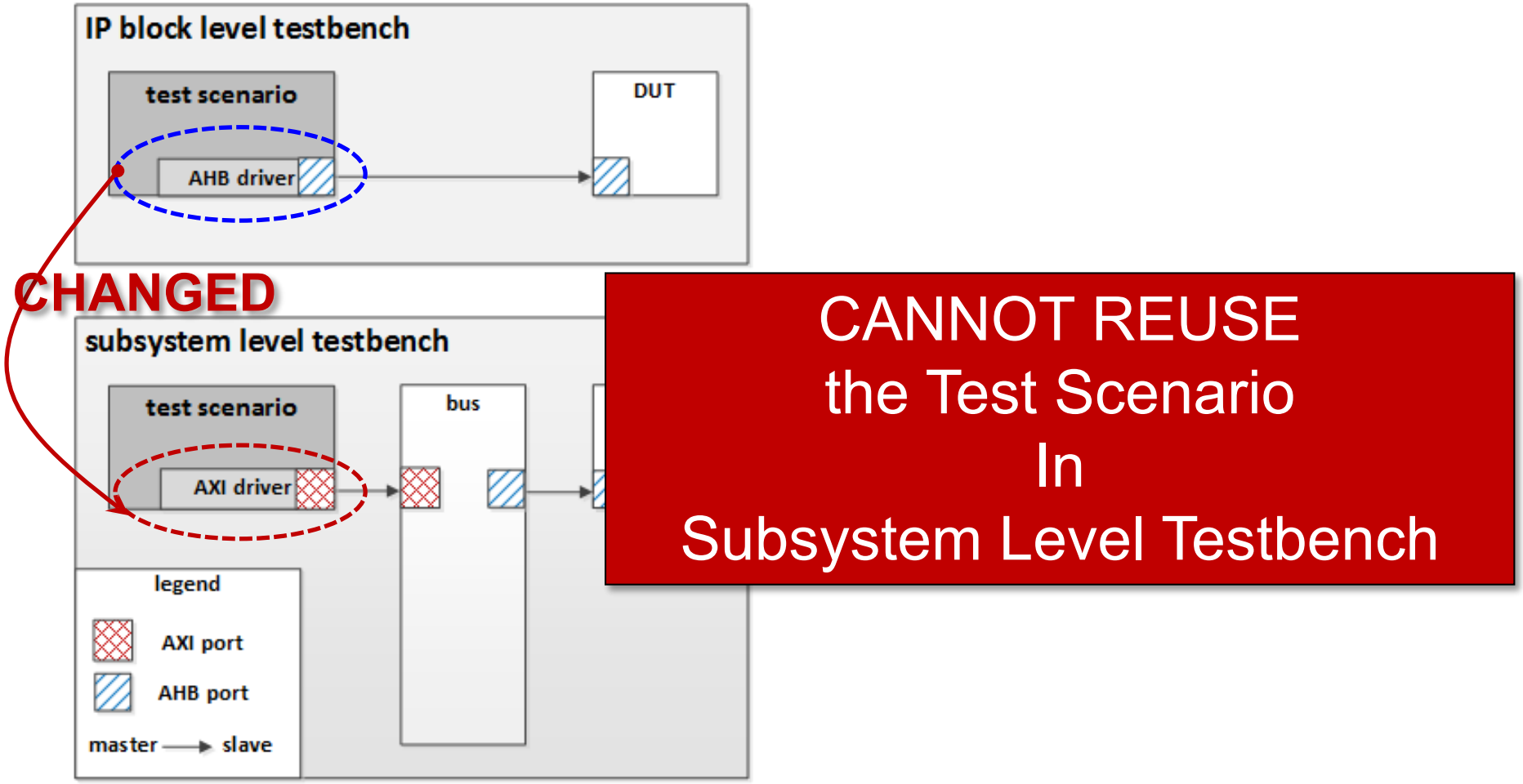
Reuse Problems : Platform Reusability

- Multiple VIP for A Test Sequence



Reuse Problems : Vertical & Horizontal Reusability

- In Case Of Changing Port Protocol



CANNOT REUSE
the Test Scenario
In
Subsystem Level Testbench

Reuse Problems : Proposed Solutions

Problem 1 : Testbench Build Time ⇨ Predefined & Configurable UVM Environment & Components

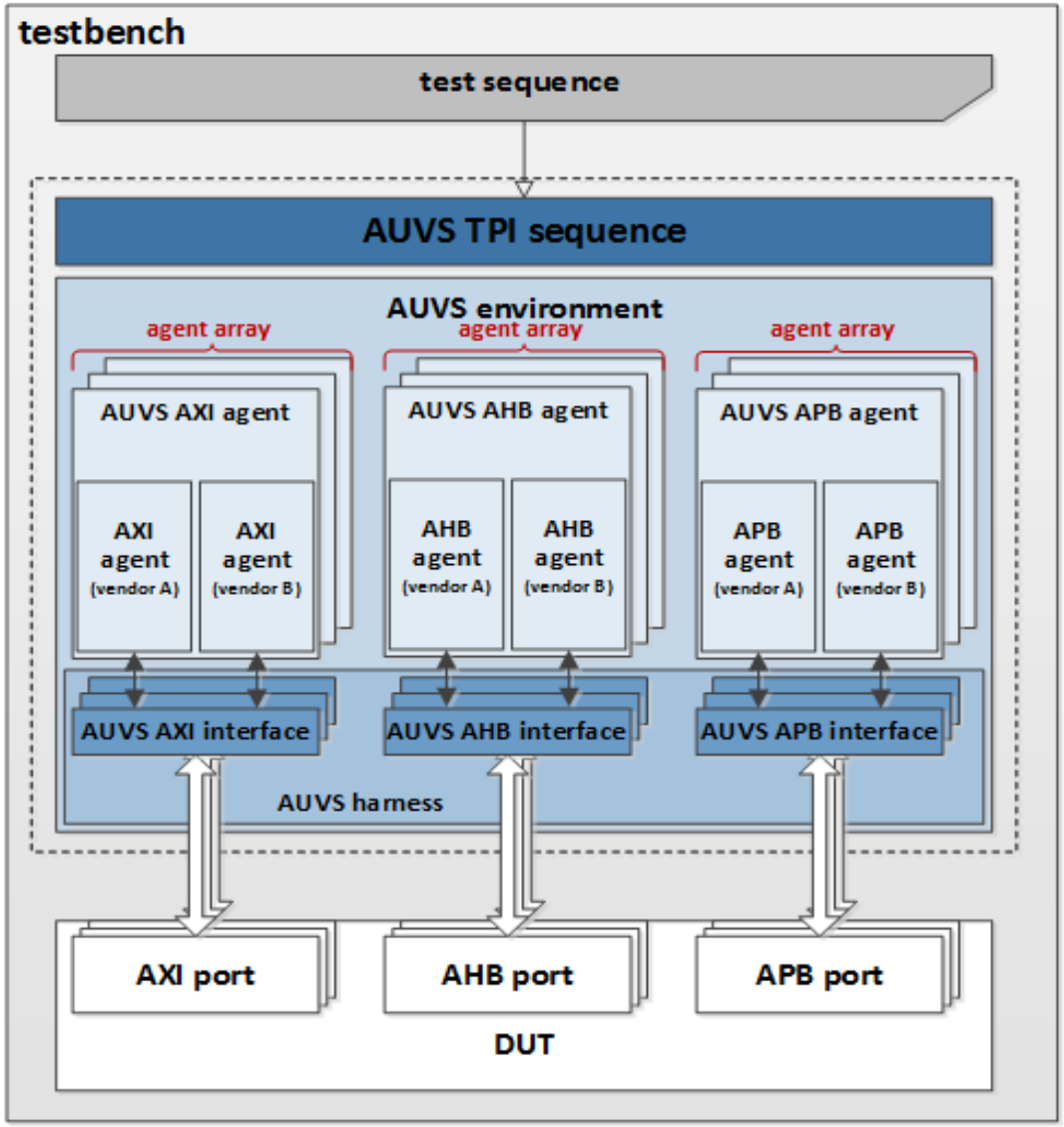
Problem 2 : Platform Reusability ⇨ Abstracted AMBA API & Layered Agent Architecture

Problem 3 : Vertical & Horizontal Reusability ⇨ Hardware Abstraction & Callbacks for Protocol Dependent Tasks

AUVS : Architecture

- AUVS
  - AMBA Unified Verification System

  1. A set of *predefined* & *configurable* AMBA VIP
  2. TPI : Test sequence Programming Interface
  3. Layered agent architecture



AUVS : Predefined System

- Configurable System
  - Number of agents
  - Type of agents : AXI/AHB/APB and Master/Slave
  - Protocol parameters of each agent
    - Address/data width
    - Monitor enable/message verbosity
    - …
- Select Target AMBA VIP in Compile Time(*Not in Code*)
  - Synopsys SVT? Questa QVIP? or In-house VIP?

AUVS : TPI

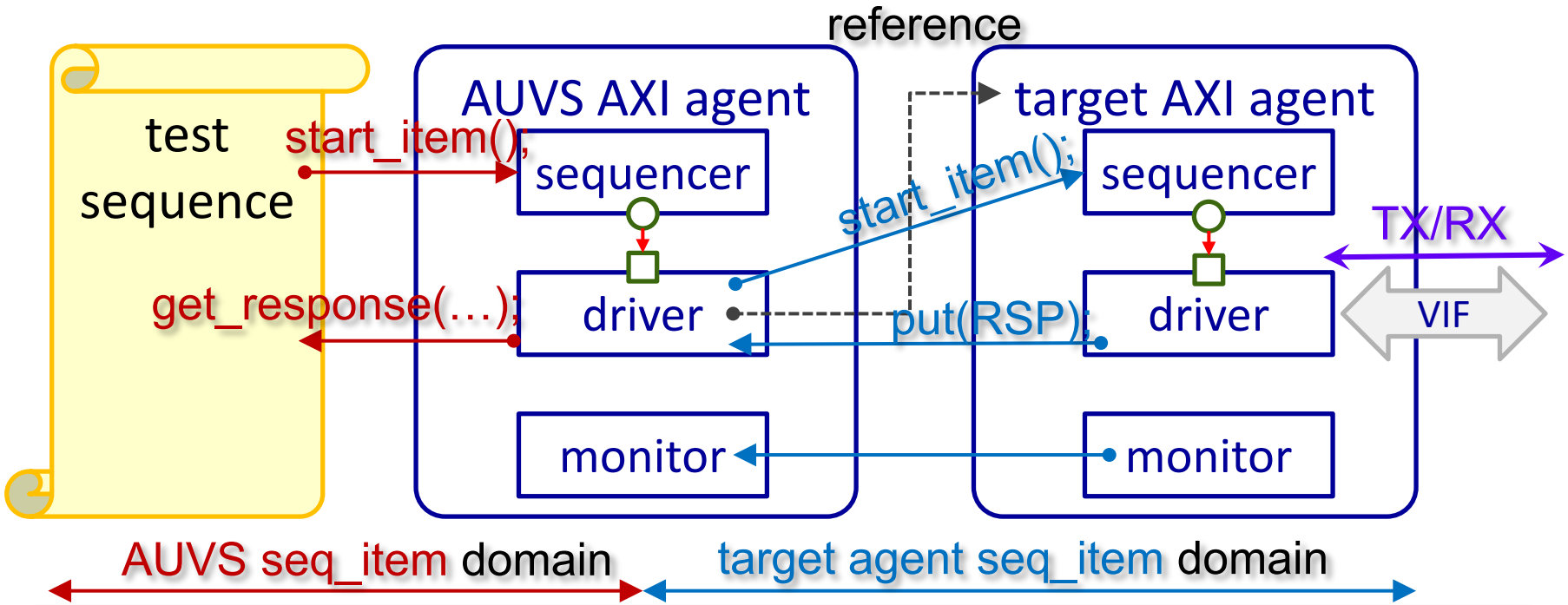
- TPI : Sets of AMBA API Tasks/Functions
  - Provides write/read tasks
  - VIP vendor independent

```
class auvs_tpi_sequence extends uvm_sequence ;
  extern task axi_write(int agt_id, bit[] axi_id, bit[]
addr,bit[] burst, bit[] size, bit[] data);
  extern task axi_read(int agt_id, bit[] axi_id, bit[]
addr,bit[] burst, bit[] size, output bit[] data);
  extern task axi_get_resp();
  extern task axi_get_resp_with_rdata();
  //...
endclass
```

Tasks Abstracting Bus Traffic

AUVS : Layered Agents

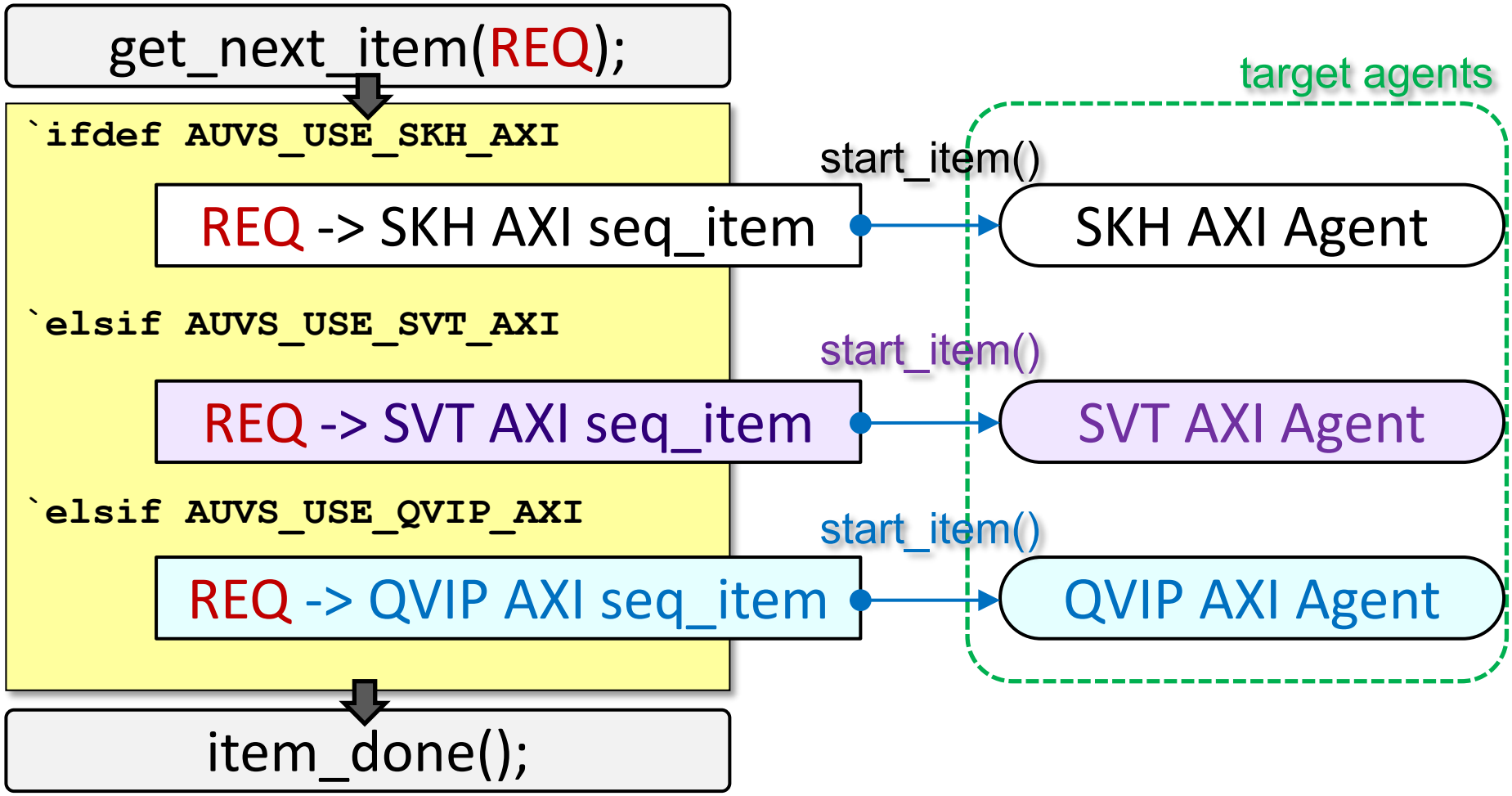
- Layered Agents for VIP Independency



Test Sequences Are Described in AUVS seq_item
Independent of Target VIP

AUVS : Layered Agents

- AUVS Drivers



AUVS : An Example

- Verification Env. Using AUVS

```
`define AUVS_USE_SKH_AXI
`define AUVS_AXI_MST_NUM 4
`define AUVS_AXI_SLV_NUM 2
`define AUVS_AHB_MST_NUM 2

class env extends uvm_env;
  function void build_phase(…);
    auvs = auvs_env::type_id::create(…);
    for ( i=0;i<AUVS_AXI_MST_NUM;i++)
      uvm_config_db#(axi_vif)::set(this,
        "auvs",
        $sformatf("axi_mst_vif[%0d]",i),
        axi_mst_vif[i]);
    //…
  endfunction
```

1. Set defines
2. Create AUVS
3. Config & Set VIFs

AUVS : An Example

- Test Sequence Using AUVS TPI

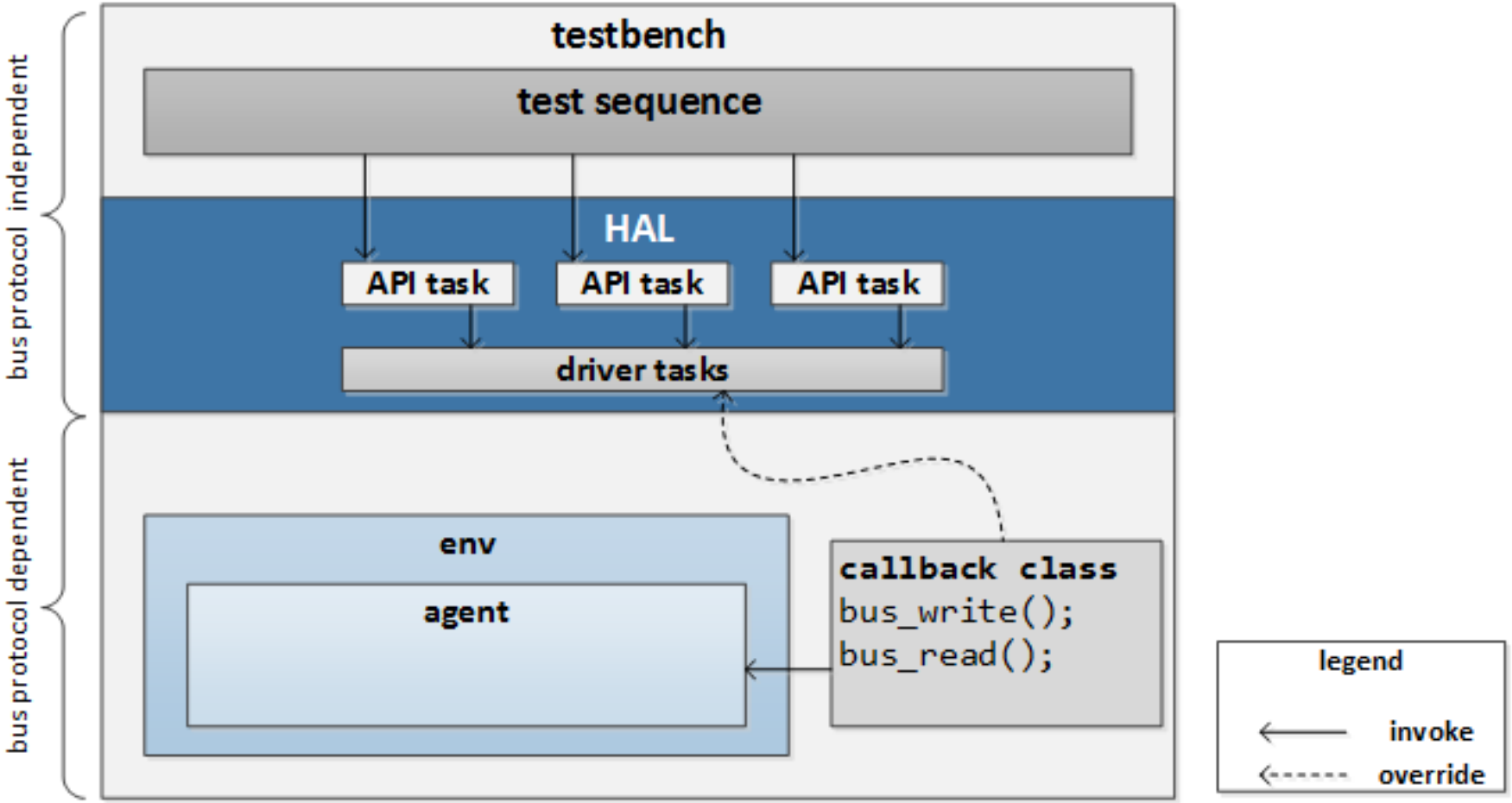
```
class auvs_sample_sequence extends auvs_tpi_sequence ;
`uvm_object_utils( auvs_sample_sequence ) ;

task body();
    int agt_id
    axi_write( agt_id=0, axi_id=0, 'h4000_0000,
               AUVS_AXI_BYTE_8, len=15, wdata,wstrb ) ;
    axi_read(  agt_id=0, axi_id=1 , 'h4000_0000 ,
               AUVS_AXI_BYTE_8 , len=15 , rdata ) ;
    ahb_write( agt_id=0, 'h5000_0000 , AUVS_AHB_INCR16 ,
                         AUVS_AHB_BYTE_4 , wdata ) ;
    ahb_read(  agt_id=0, 'h5000_0000 , AUVS_AHB_INCR16 ,
                         AUVS_AHB_BYTE_4 , rdata ) ;
  endtask
endclass
```

TPI Tasks
There Is No VIP Dependent Information

FLS : Architecture

- Firmware-Like Architecture Using HAL



FLS : Test Sequence

- Why “Firmware-Like” ?

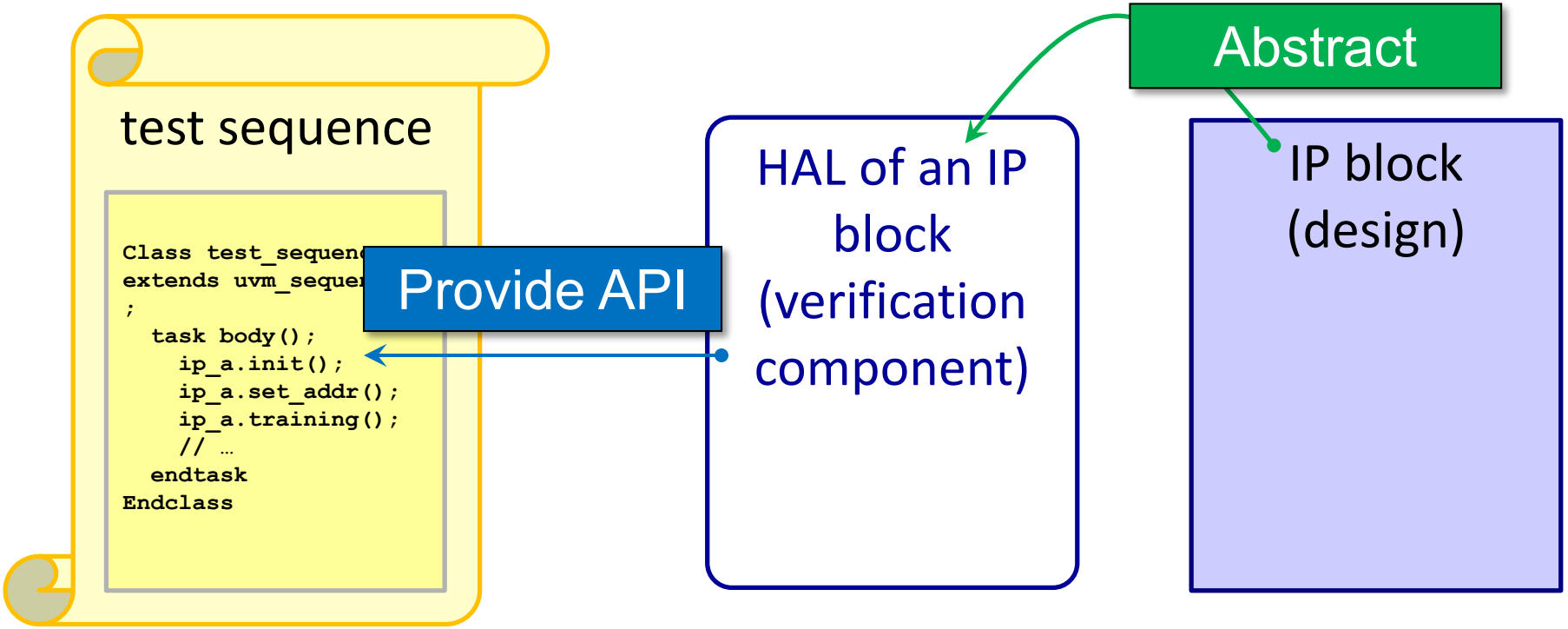
```
class ddrc_seq extends uvm_sequence ;
ddrc_hal ddrc ; // DDR Controller HAL

task body() ;
    ddrc.check_reset() ;
    ddrc.set_addr(row=15,col=11) ;
    ddrc.init_mc();
    ddrc.training_enable();
    ddrc.init_phy();
    while ( !ddrc.is_init_mc_done() ) ;
    ddrc.get_status(rdata);
    // …
    ddrc.set_self_refresh_mode();
    ddrc.get_status(rdata);
endtask
```

Describe Test Sequence ONLY using HAL API tasks

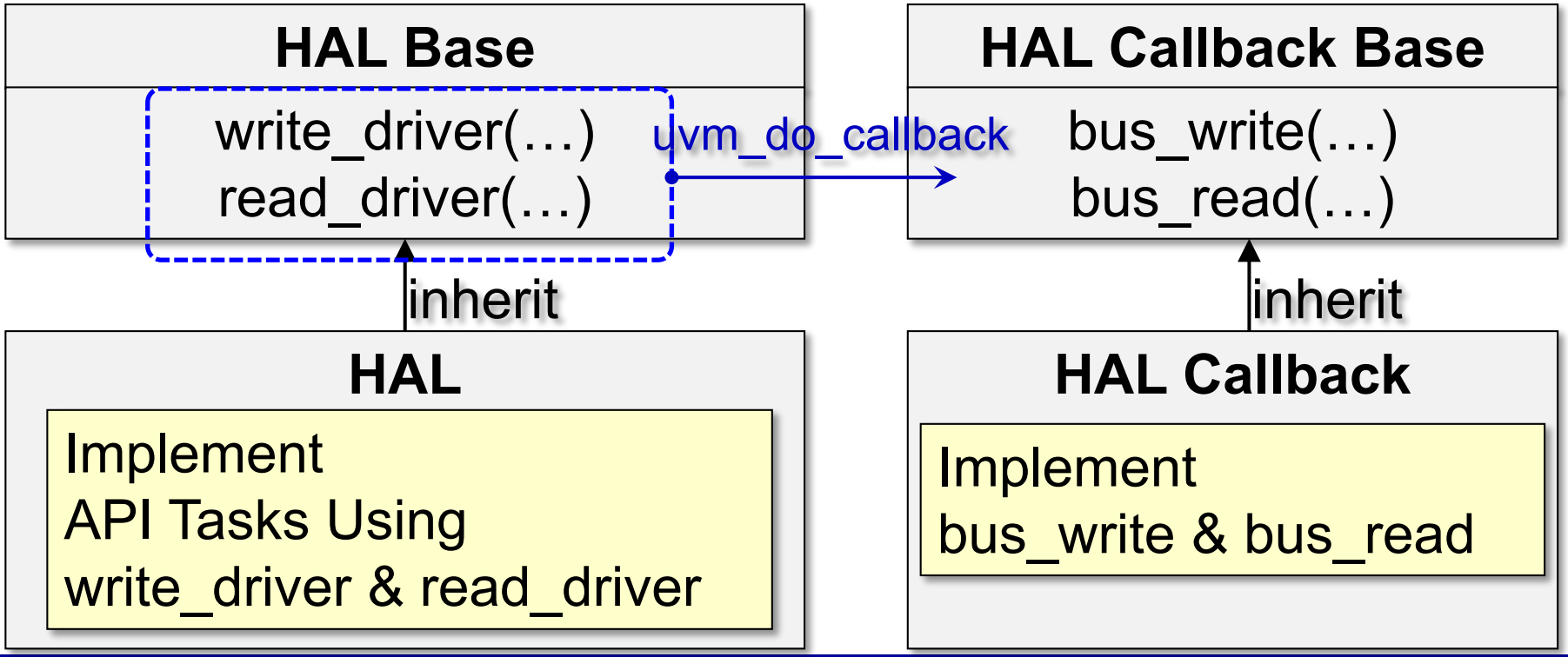
FLS : HAL

- HAL : Hardware Abstraction Layer
  - Abstract functions of the IP block
  - Provide test sequences with HAL API tasks/functions



FLS : HAL Base

- HAL Base & HAL Callback Base
  - Base Class for HAL.
  - Provide Abstracted Bus Traffic Tasks.



FLS : Bus Protocol Independent

- Using '*uvm_callback*' : HAL Base

Register '*hal_callback_base*' as Callback Class

```
class hal_base extends uvm_obje
  `uvm_register_cb( hal_base , hal_callback_base ) ;

task write_driver( bit[31:0] addr , bit[31:0] data ) ;

  `uvm_do_callback( hal_base,hal_callback_base
,bus_write(addr,data));

endtask
endclass
```

Use `*uvm_do_callback* to Invoke Callback Task

FLS : Bus Protocol Independent

- Using '*uvm_callback*' : HAL Callback Base

```
class hal_callback_base extends uvm_callback ;
      // virtual task. Real operation isn't implemented here.
      virtual task bus_write(bit[] addr, bit[] data);
      endtask

      virtual task bus_read(bit[] addr, output bit[] data);
      endtask
endclass
```

These Tasks Are Expected to BE IMPLEMENTED with a Specific AMBA Protocol Operation

FLS : An Example

- HAL of DDR Controller : ddrc_hal.sv

```
class ddrc_hal extends hal_base
;
task mr_write();
  write_driver( addr , data );
endtask
task mr_read();
  write_driver( addr , data );
endtask
 task set_address();
  write_driver( addr , data );
endtask
task set_phy_delay();
  write_driver( addr , data ) ;
endtask
```

```
task set_ddrc_start();
  read_driver( `DDRC_CTRL,
rdata ) ;
  rdata[0] = 1 ;
  write_driver( `DDRC_CTRL,
rdata ) ;
endtask

endclass
```

FLS : An Example

- HAL of DDR Controller : ddrc_hal_callback.sv

```
class ddrc_hal_callback extends hal_callback_base ;

task bus_write(bit[] addr, bit[] data);
  int agt_id ;
  seq.ahb_write( agt_id = 0, addr, AUVS_AHB_BURST_SINGLE,
                                   AUVS_AHB_SIZE_BYTE_4,data) ;
endtask

task bus_read(bit[] addr, output bit[] data);
  int agt_id ;
  seq.ahb_read( agt_id = 0, addr, AUVS_AHB_BURST_SINGLE,
                                  AUVS_AHB_SIZE_BYTE_4, data) ;
endtask

endclass
```

FLS : An Example

- HAL of DDR Controller : Test Sequence

```
class ddrc_sample_sequence extends auvs_tpi_sequence ;
  ddrc_hal#(auvs_tpi_sequence) ddrc ;
  ddrc_hal_callback ddrc_cb ;
task set_hal();
  ddrc = new("ddrc");
  ddrc_cb=new("ddrc_cb");
  ddrc_cb.set_seq( this );
  uvm_callbacks#(hal_base,hal_callback_base)::add(ddrc,
                                                  ddrc_cb);
endtask

task body();
    ddrc.init();
endtask

endclass
```

Conclusions

- Reuse Problems
  - Problem 1 : testbench build
  - Problem 2 : platform reusability
  - Problem 3 : vertical and horizontal reusability

These Reuse Problems Cause REWRITING of Verification Components.

Conclusions

- AMBA Unified Verification System
  - Reduce testbench build-Up time (Problem 1.)
  - Remove dependency on VIPs (Problem 2.)

- Firmware-Like Sequences
  - Abstract functions of a Design.
  - Ensure protocol independency (Problem 3.)

AUVS + FLS Improve
The Reusability of Verification Components.

Q&A