

UVM Register Use Models

Mark Glasser

Methodology Architect

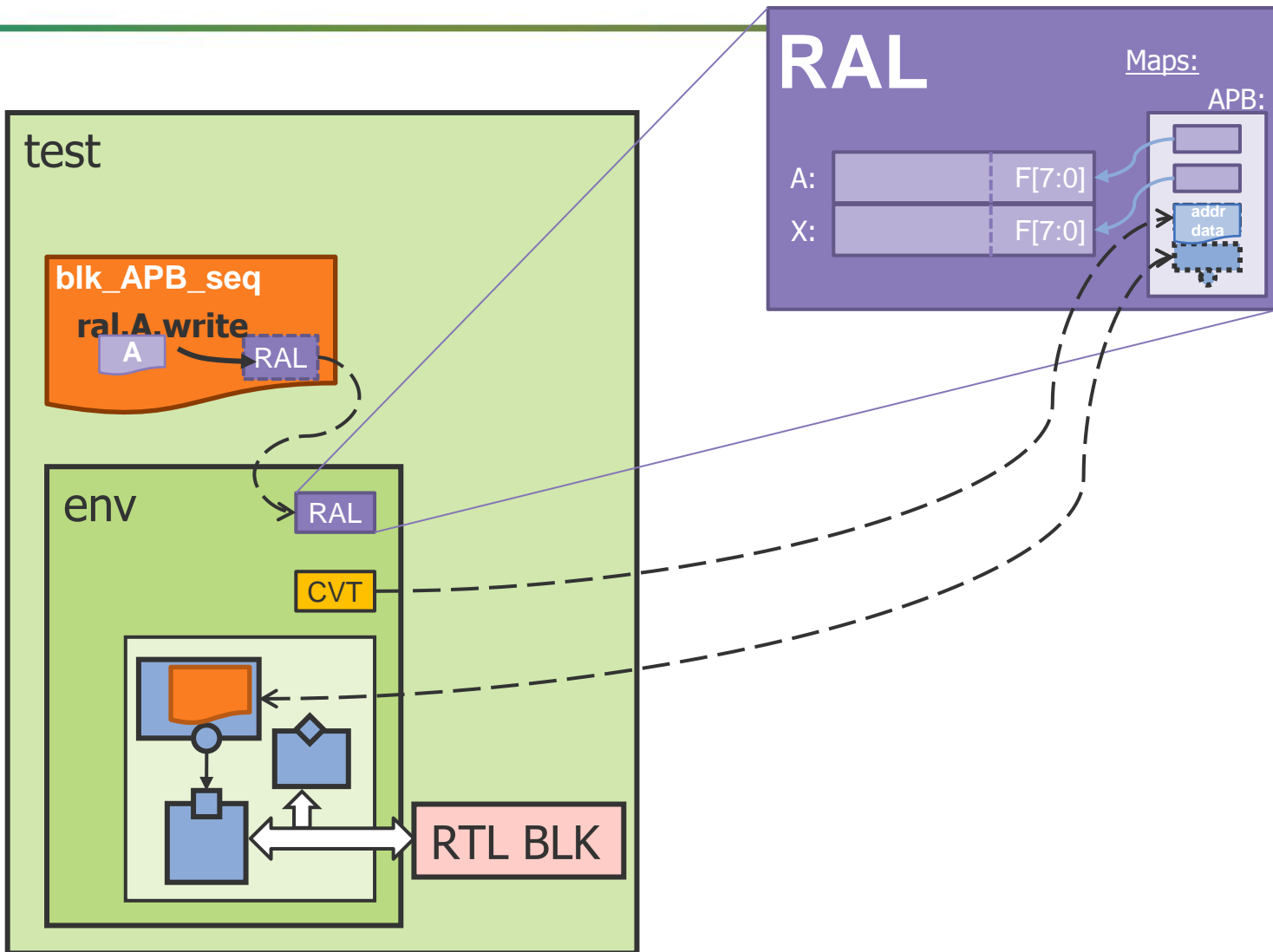
DVT

October 2010

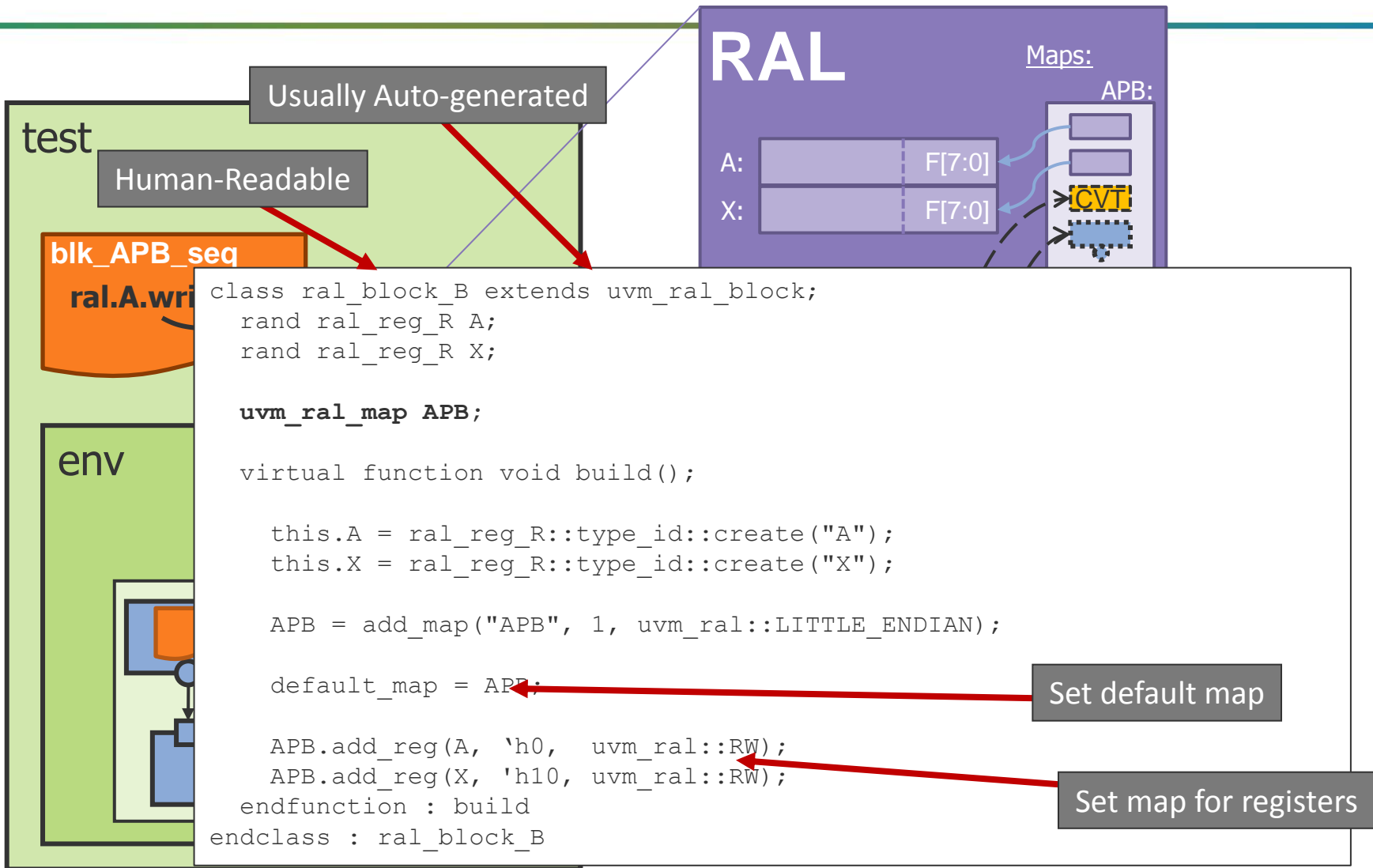
UVM Register

- SystemVerilog/UVM facility for modeling registers
- Variety of use models
 - Single bus
 - Multiple bus
 - Hierarchy
- Driven by Sequences

UVM RAL: Basic Flow



Infrastructure: The RAL Block



Step 1: The Converter

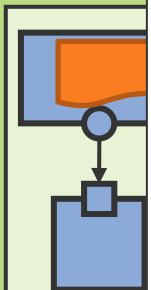
RAL

Maps:

test

blk_APB_seq
ral.A.write

env



```
class ral2apb_adapter extends uvm_ral_convert;

  `uvm_object_utils(ral2apb_adapter)

  virtual function uvm_sequence_item ral2bus(uvm_ral_item ral_item);
    apb_item apb = apb_item::type_id::create("apb_item");
    apb.read = (ral_item.oper == uvm_ral::READ) ? 1 : 0;
    apb.addr = ral_item.addr;
    apb.data = ral_item.data;
    return apb;
  endfunction

  virtual function void bus2ral(uvm_sequence_item bus_item,
                                uvm_ral_item ral_item);

    apb_item apb;
    if (!$cast(apb, bus_item)) begin
      `uvm_fatal("NOT_APB_TYPE", "Provided bus_item is the wrong type")
      return;
    end
    ral_item.oper = apb.read ? uvm_ral::READ : uvm_ral::WRITE;
    ral_item.addr = apb.addr;
    ral_item.data = apb.data;
  endfunction
endclass
```

Not called directly
by user

Step 2: The RAL Sequence

RAL

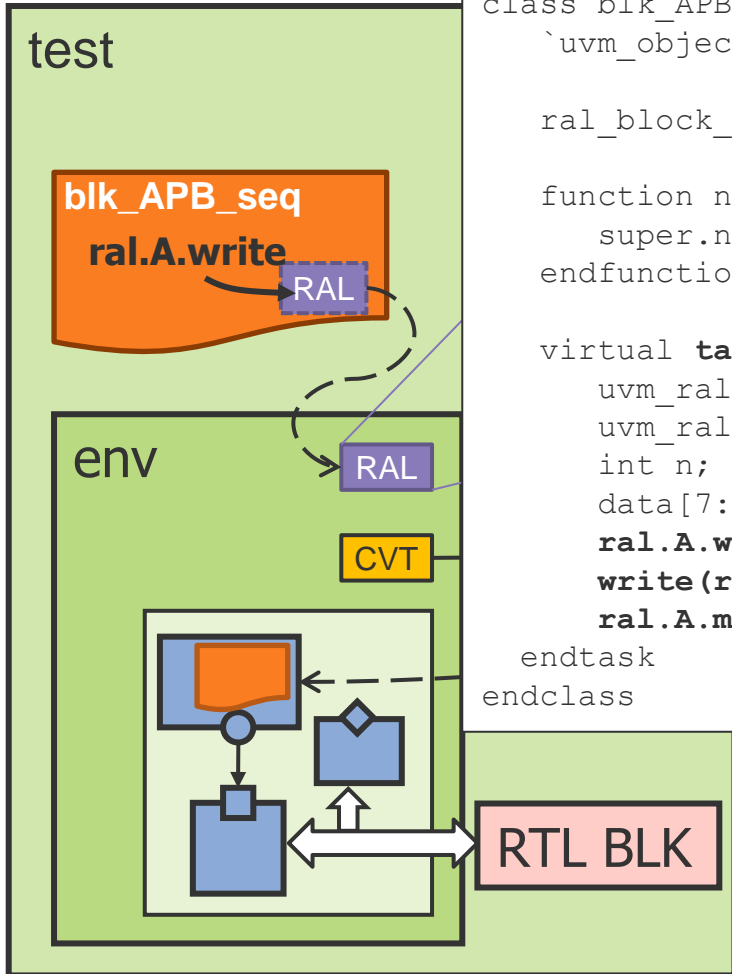
Maps:

```
class blk_APB_test_seq extends uvm_ral_sequence;
  `uvm_object_utils(blk_AXW_test_seq)

  ral_block_B ral;

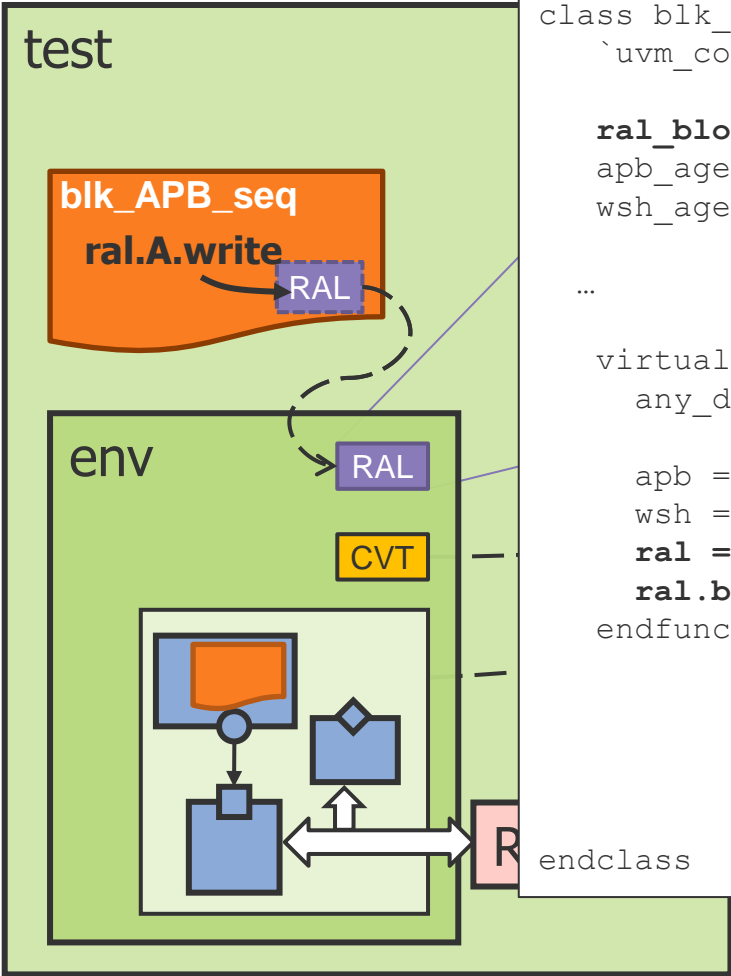
  function new(string name = "blk_AXW_test_seq");
    super.new(name);
  endfunction: new

  virtual task body();
    uvm_ral::status_e status;
    uvm_ral_data_t data;
    int n;
    data[7:0] = $urandom();
    ral.A.write(status, data, .parent(this));
    write(ral.X, status, data);
    ral.A.mirror(status, uvm_ral::VERB, .parent(this));
  endtask
endclass
```



Step 3: The Environment

RAL Maps: APB



```

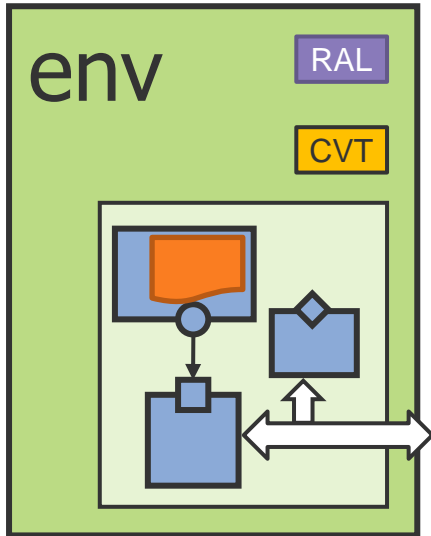
class blk_env extends uvm_env;
  `uvm_component_utils(blk_env)

  ral_block_B  ral;
  apb_agent    apb;
  wsh_agent    wsh;

  ...

  virtual function void build();
    any_driver#(apb_item)::type_id::set_type_override(
      my_apb_driver::get_type());
    apb = apb_agent::type_id::create("apb_agent",this);
    wsh = wsh_agent::type_id::create("wsh_agent",this);
    ral = ral_block_B::type_id::create("ral_blk_B");
    ral.build();
  endfunction: build
endclass
  
```

Step 4: Pass Sequencer to RAL Block



```
class blk_env extends uvm_env;
  `uvm_component_utils(blk_env)

  ral_block_B  ral;
  apb_agent    apb;
  wsh_agent    wsh;

  ...

  virtual function void build();
    any_driver#(apb_item)::type_id::set_type_override(
        my_apb_driver::get_type());
    apb =  apb_agent::type_id::create("apb_agent",this);
    wsh =  wsh_agent::type_id::create("wsh_agent",this);
    ral =  ral_block_B::type_id::create("ral_blk_B");
    ral.build();
  endfunction: build

  virtual function void connect();
    ral2apb_adapter ral2apb = new;
    ral.APB.set_sequence(apb.sqr, ral2apb);
  endfunction
endclass
```

Step 5: The Test

RAI

Maps:

test

blk_APB_seq
ral.A.write

RAL

env

RA

CV

endclass

RTL BLK

```
class blk_test extends uvm_test;
  `uvm_component_utils(blk_test)

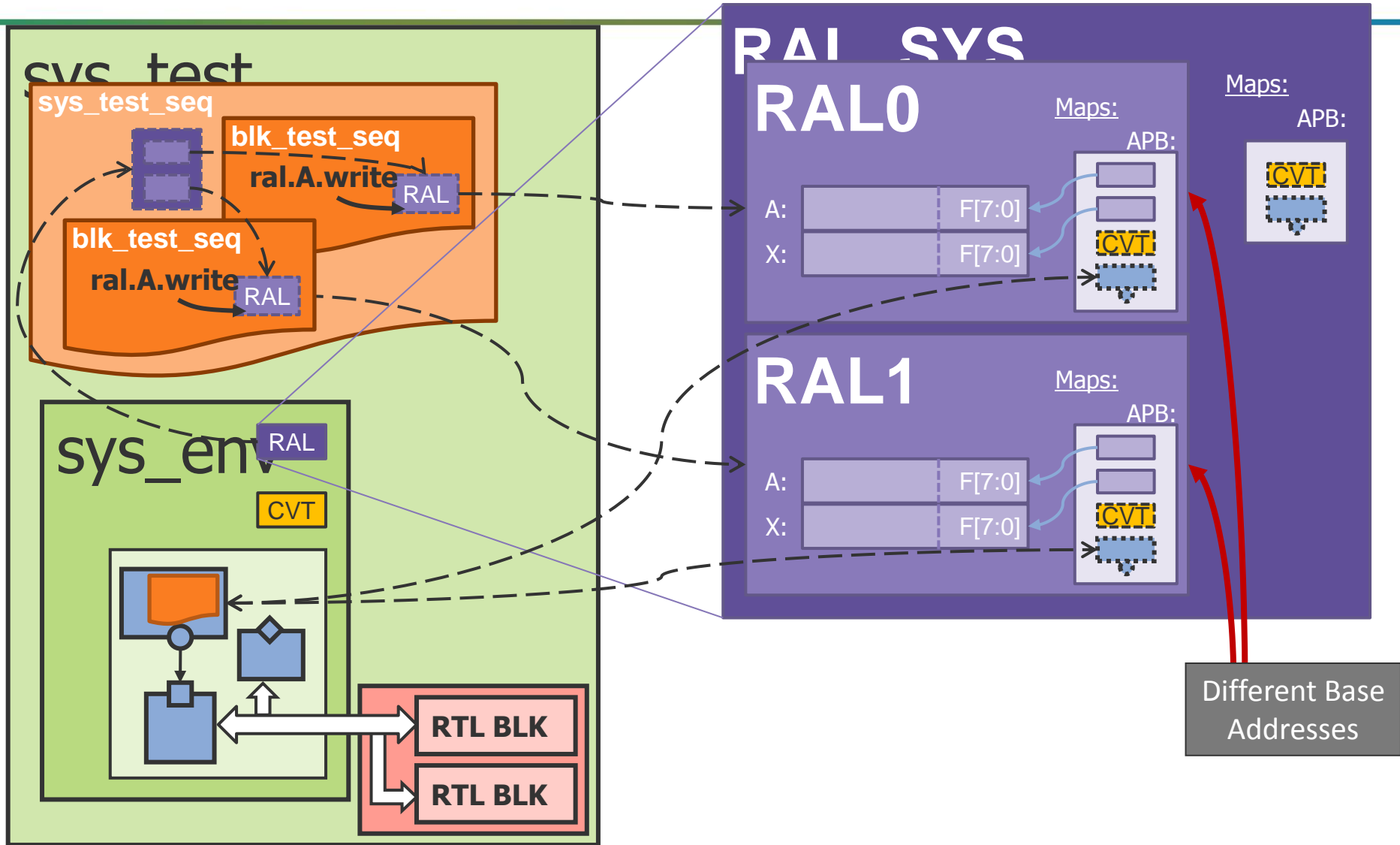
  blk_env env;
  ...
  function void build();
    env = blk_env::type_id::create("blk_env",this);
  endfunction

  task run();
    blk_APB_seq seq = blk_APB_seq::type_id::create("blk_APB_seq",this);
    seq.ral = env.ral;

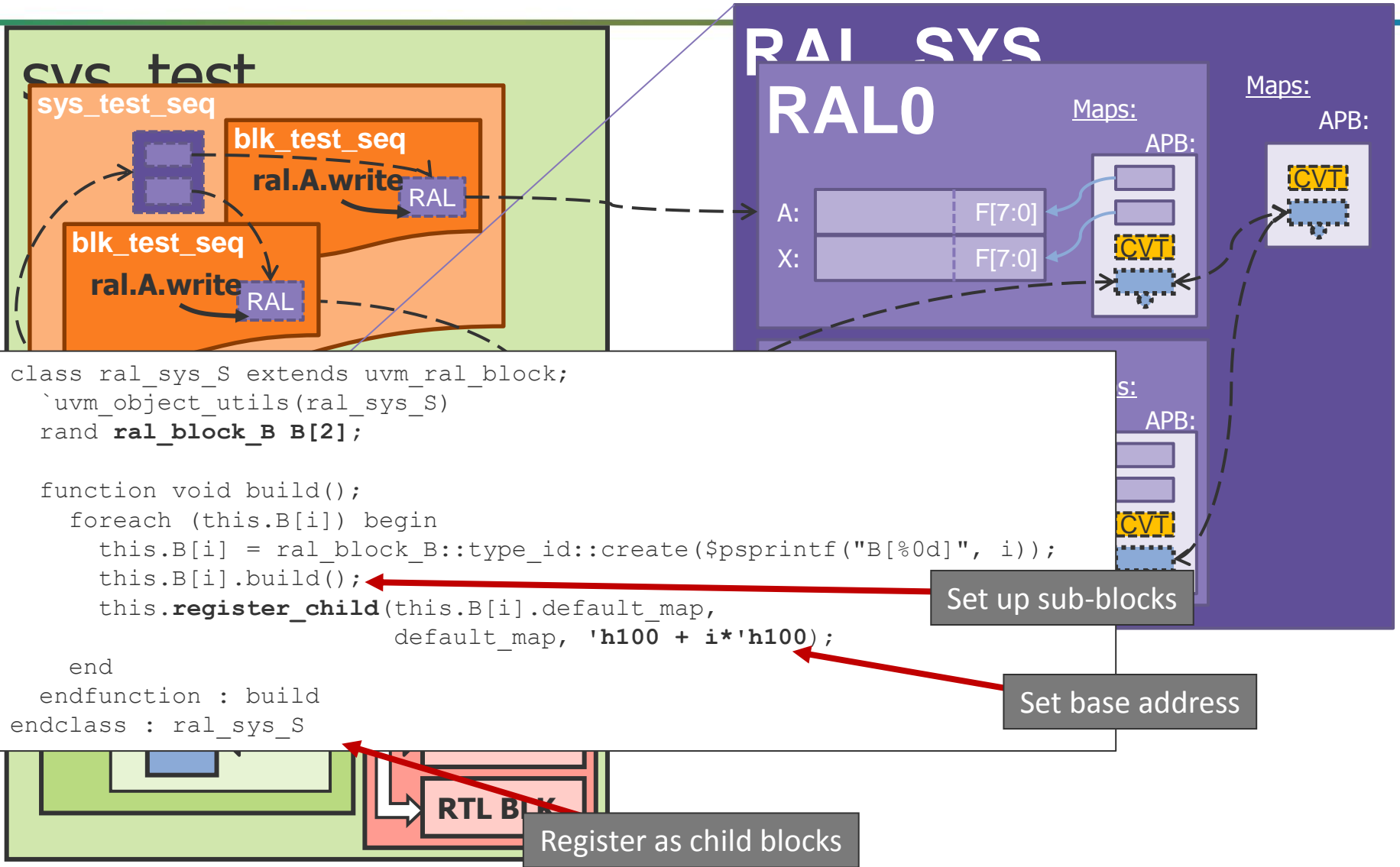
    seq.start(null);
    seq.wait_for_sequence_state(FINISHED);

    global_stop_request();
  endtask
endclass
```

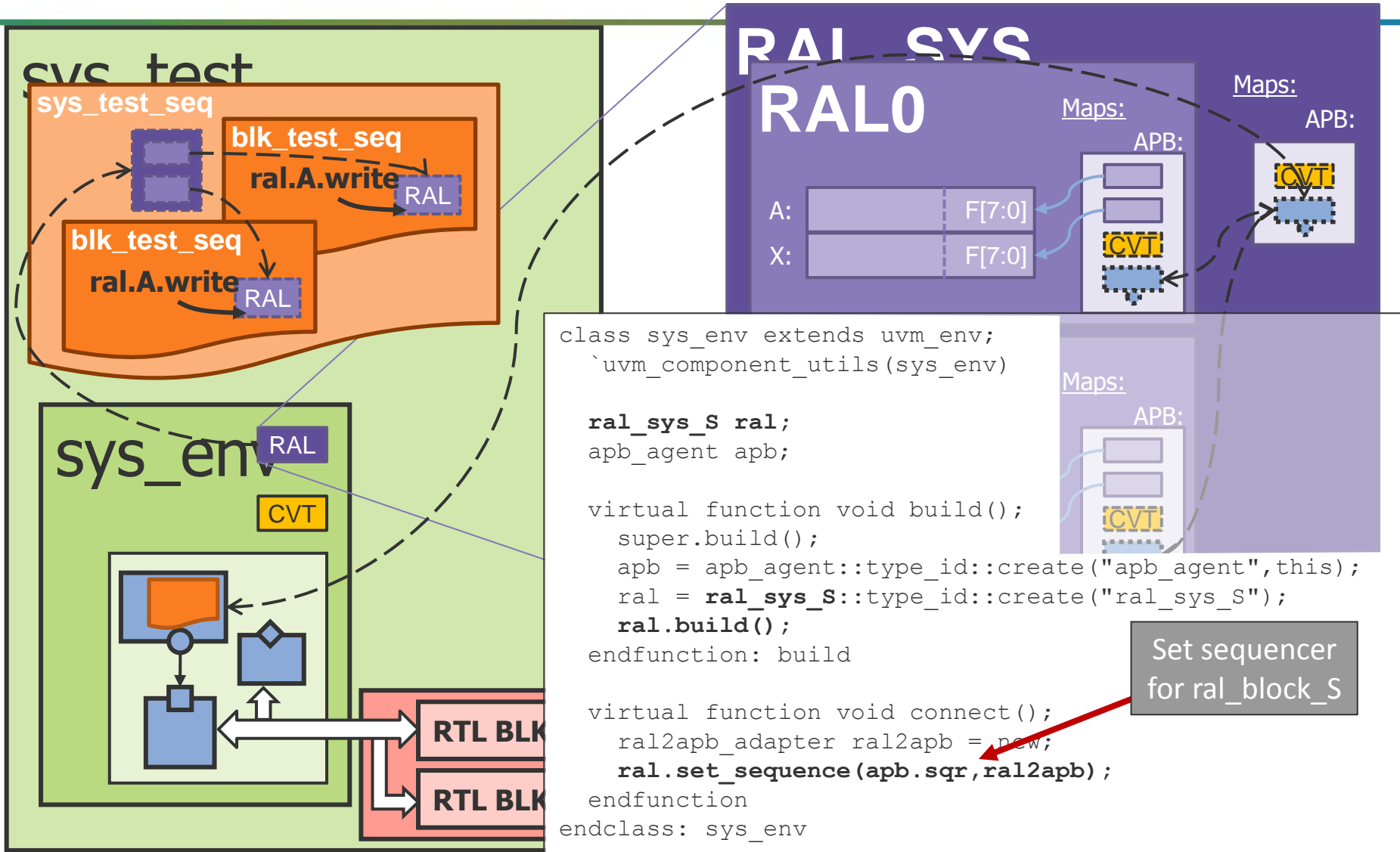
Example: Vertical Reuse



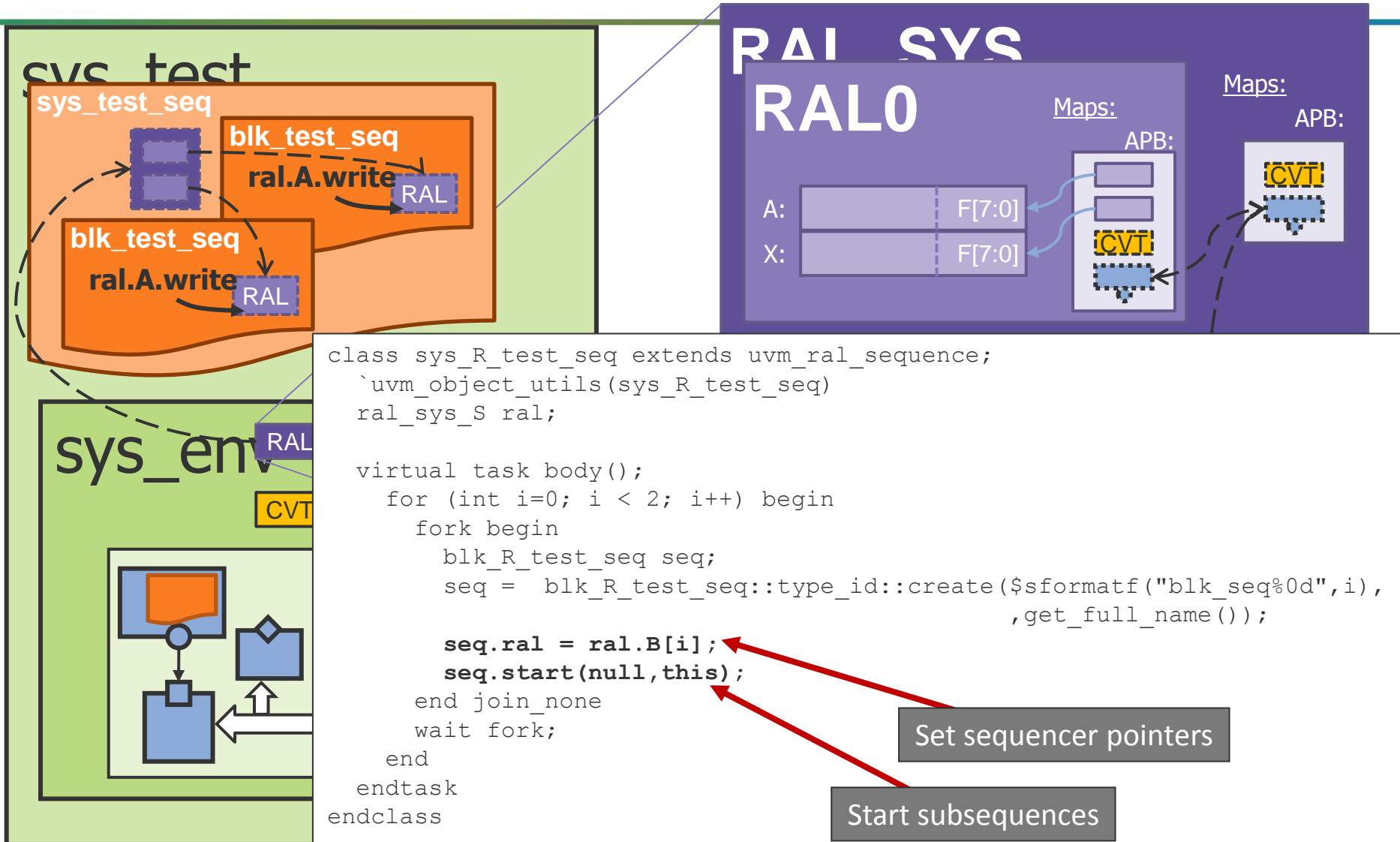
Vertical Reuse: The RAL Block



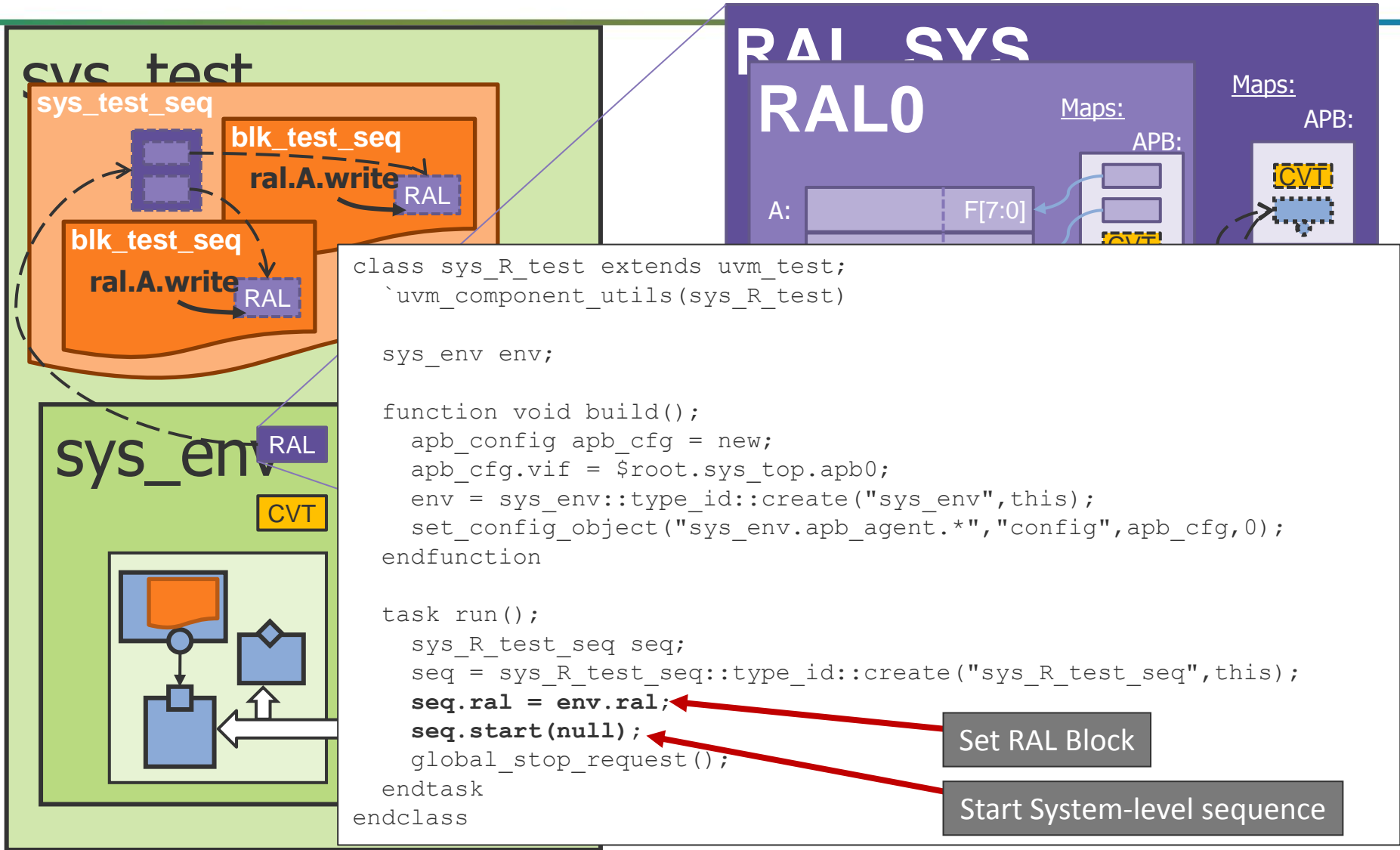
Vertical Reuse: The Environment



Vertical Reuse: The Test Sequence



Vertical Reuse: The Test



```

class sys_R_test extends uvm_test;
  `uvm_component_utils(sys_R_test)

  sys_env env;

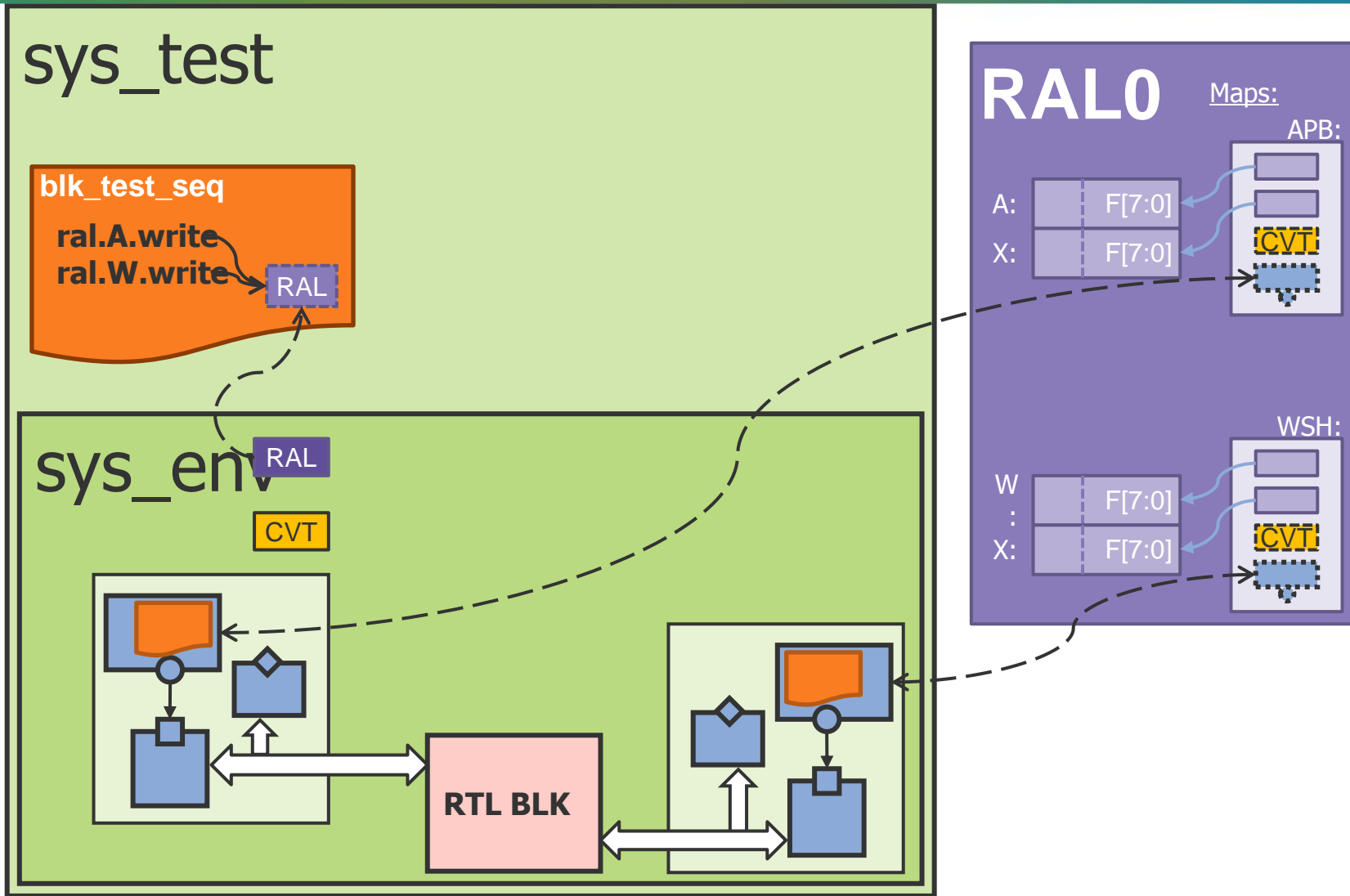
  function void build();
    apb_config apb_cfg = new;
    apb_cfg.vif = $root.sys_top.apb0;
    env = sys_env::type_id::create("sys_env",this);
    set_config_object("sys_env.apb_agent.*", "config", apb_cfg, 0);
  endfunction

  task run();
    sys_R_test_seq seq;
    seq = sys_R_test_seq::type_id::create("sys_R_test_seq",this);
    seq.ral = env.ral;
    seq.start(null);
    global_stop_request();
  endtask
endclass
  
```

Set RAL Block

Start System-level sequence

Example 3: Multiple Maps



Multiple Maps: The RAL Block

sys_test

```
class ral_block_B extends uvm_ral_block;  
  rand ral_reg_R A;  
  rand ral_reg_R X;  
  rand ral_reg_R W;  
  
  uvm_ral_map APB;  
  uvm_ral_map WSH;  
  
  virtual function void build();  
    this.A = ral_reg_R::type_id::create("A");  
    this.X = ral_reg_R::type_id::create("X");  
    this.W = ral_reg_R::type_id::create("W");  
    APB = add_map("APB", 1, uvm_ral::LITTLE_ENDIAN);  
    WSH = add_map("WSH", 1, uvm_ral::LITTLE_ENDIAN);  
    default_map = APB;  
  
    APB.add_reg(A, 'h0, uvm_ral::RW);  
    APB.add_reg(X, 'h10, uvm_ral::RW);  
    WSH.add_reg(X, 'h0, uvm_ral::RW);  
    WSH.add_reg(W, 'h10, uvm_ral::RW);  
  endfunction  
endclass
```

blk_t
ral.
ral.

Declare Maps

Add Maps

Register shared in 2 maps



Multiple Maps: The Sequence

sys_test

```

class blk_test_seq extends uvm_ral_sequence;
  `uvm_object_utils(blk_AXW_test_seq)

  ral_block_B ral;

  virtual task body();
    uvm_ral::status_e status;
    uvm_ral_data_t data;
    fork
      ral.A.write(status, 'h33, .parent(this));
      ral.X.write(status, 'h44, .parent(this));
      ral.W.write(status, 'hcc, .parent(this));
    join

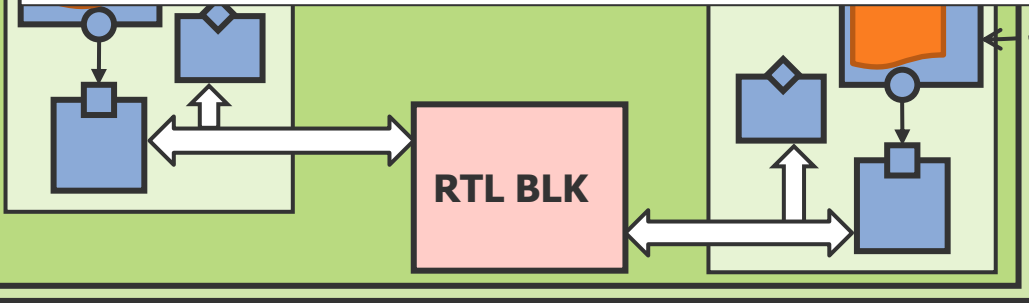
    ral.X.write(status, ~ral.A.get(), .map(ral.APB), .parent(this));
    ral.X.mirror(status, uvm_ral::VERB, .map(ral.WSH), .parent(this));
  endtask
endclass
  
```

Parallel accesses via default maps

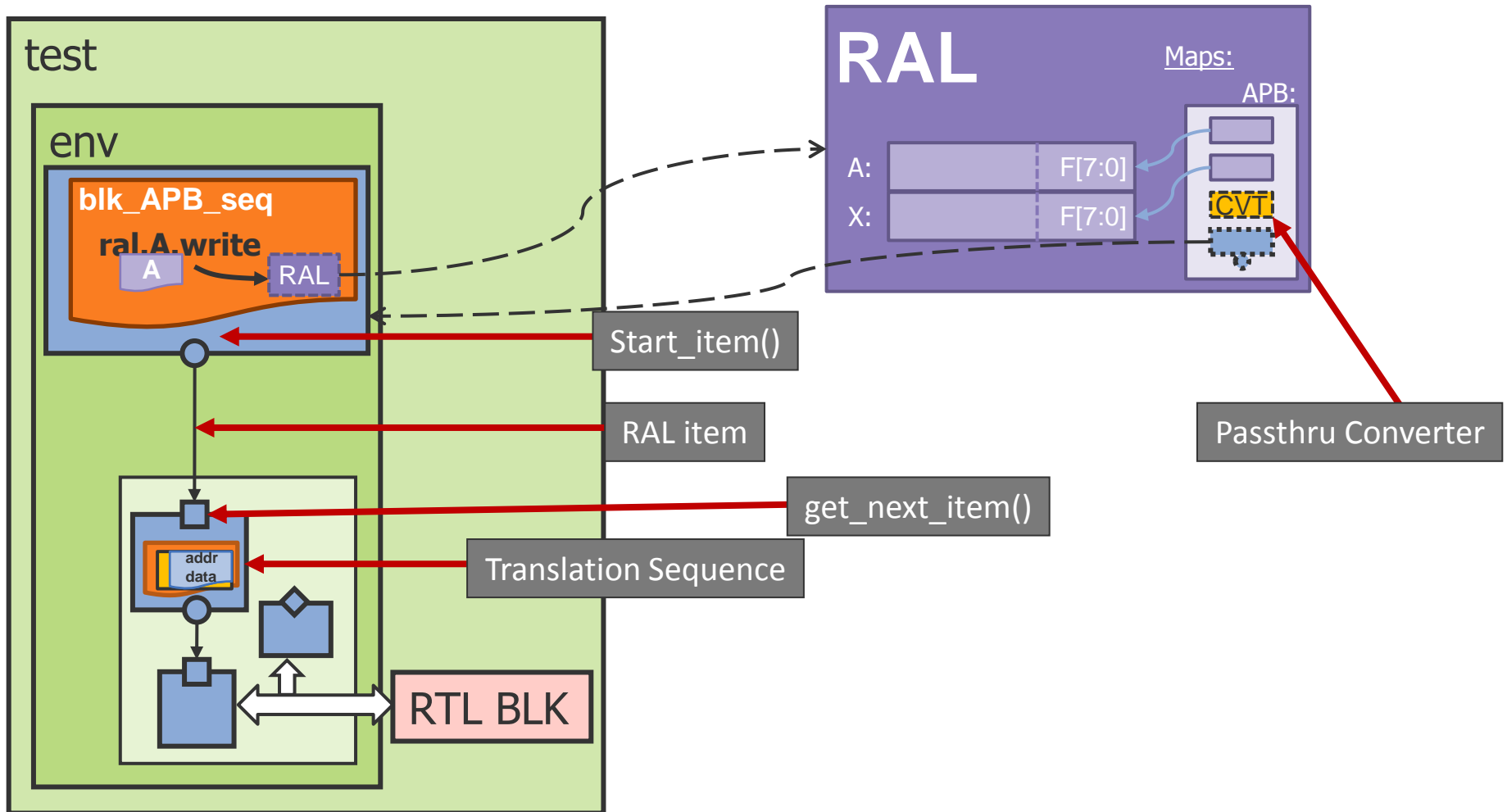
Explicit Map

One register accessed via multiple maps

sys



UVM RAL: Translation Sequence



Thank You!

**Mentor
Graphics®**

www.mentor.com