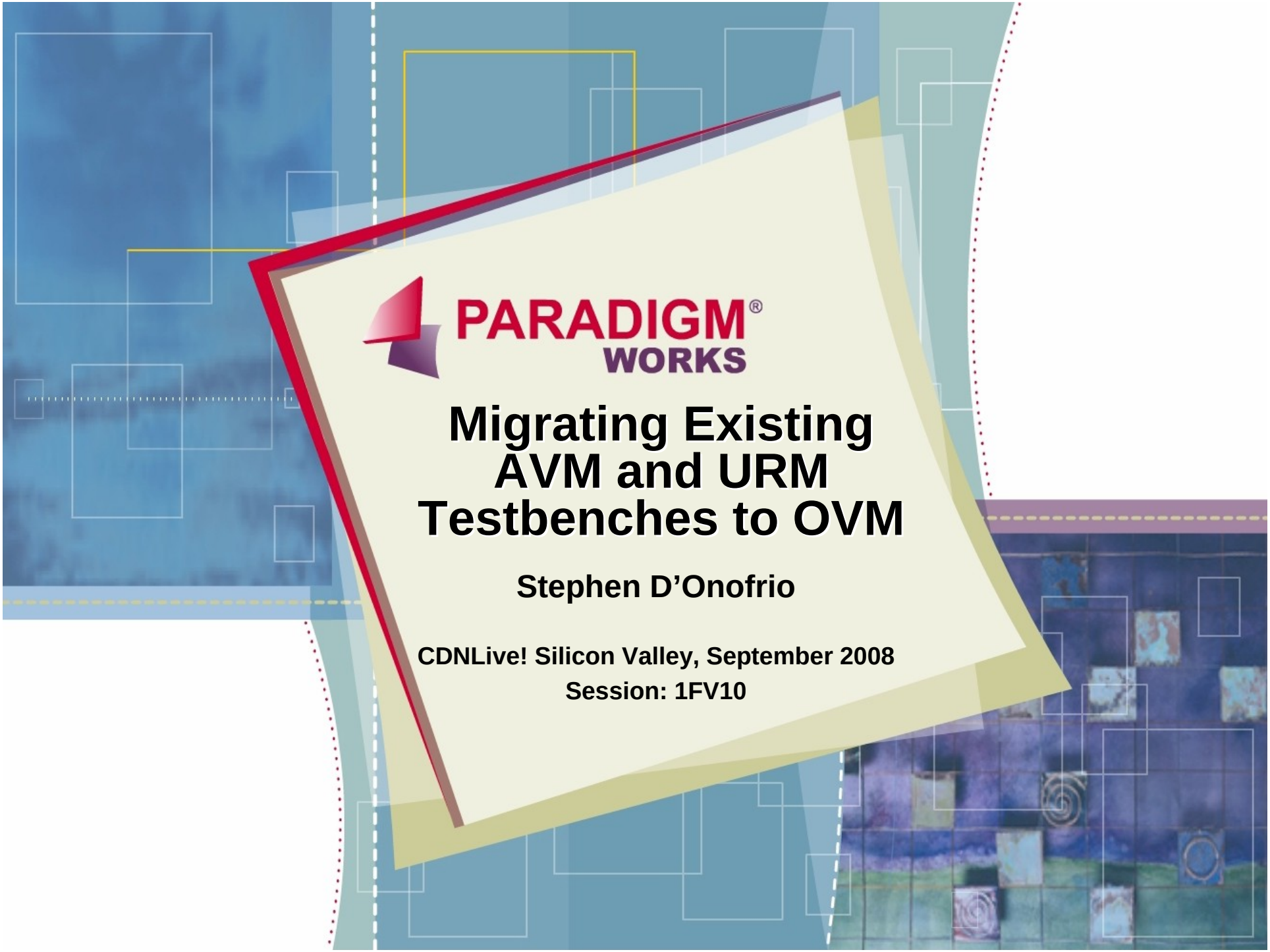


Migrating Existing AVM and URM Testbenches to OVM

Stephen D'Onofrio

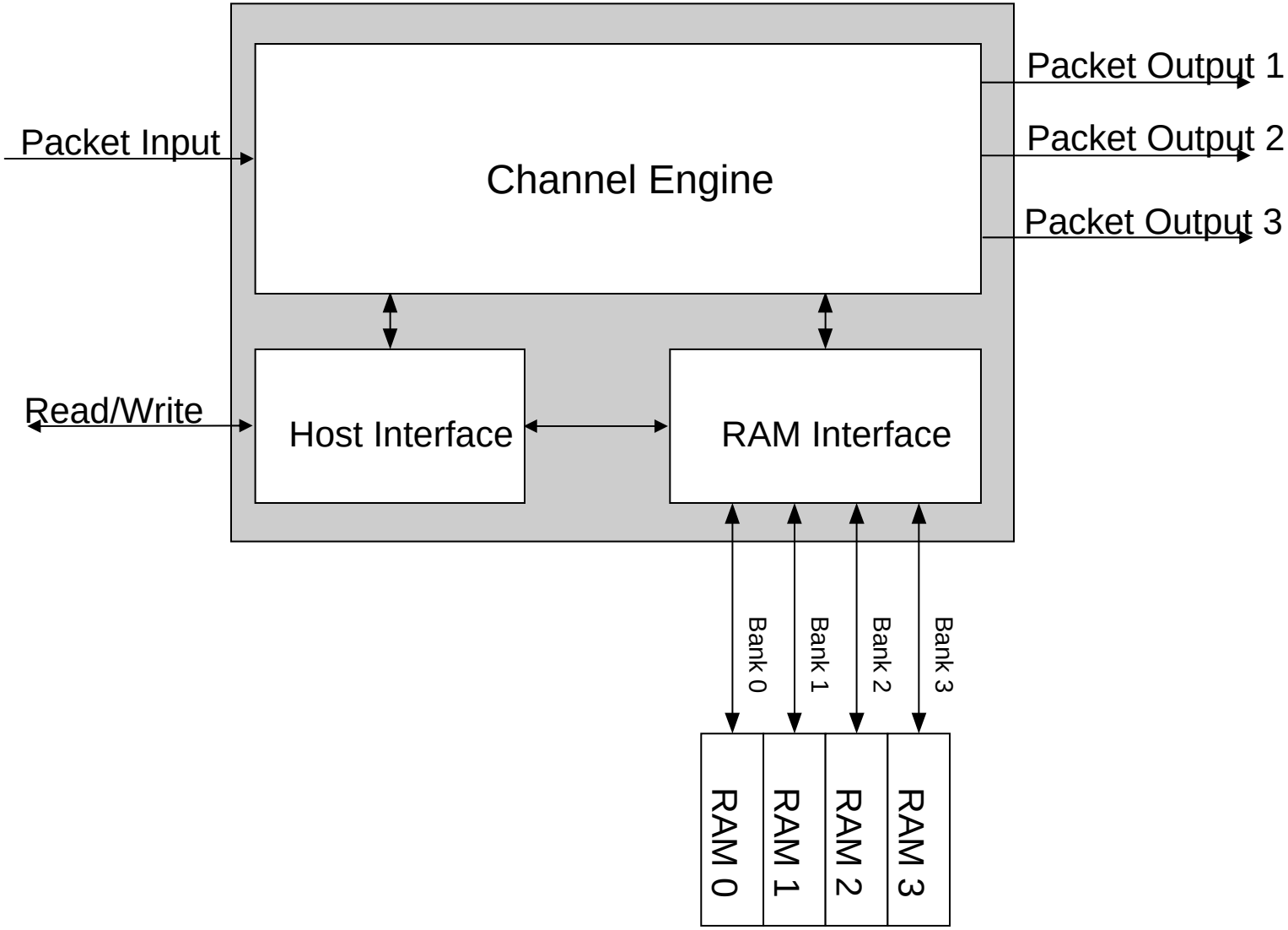
CDNLive! Silicon Valley, September 2008
Session: 1FV10

Outline

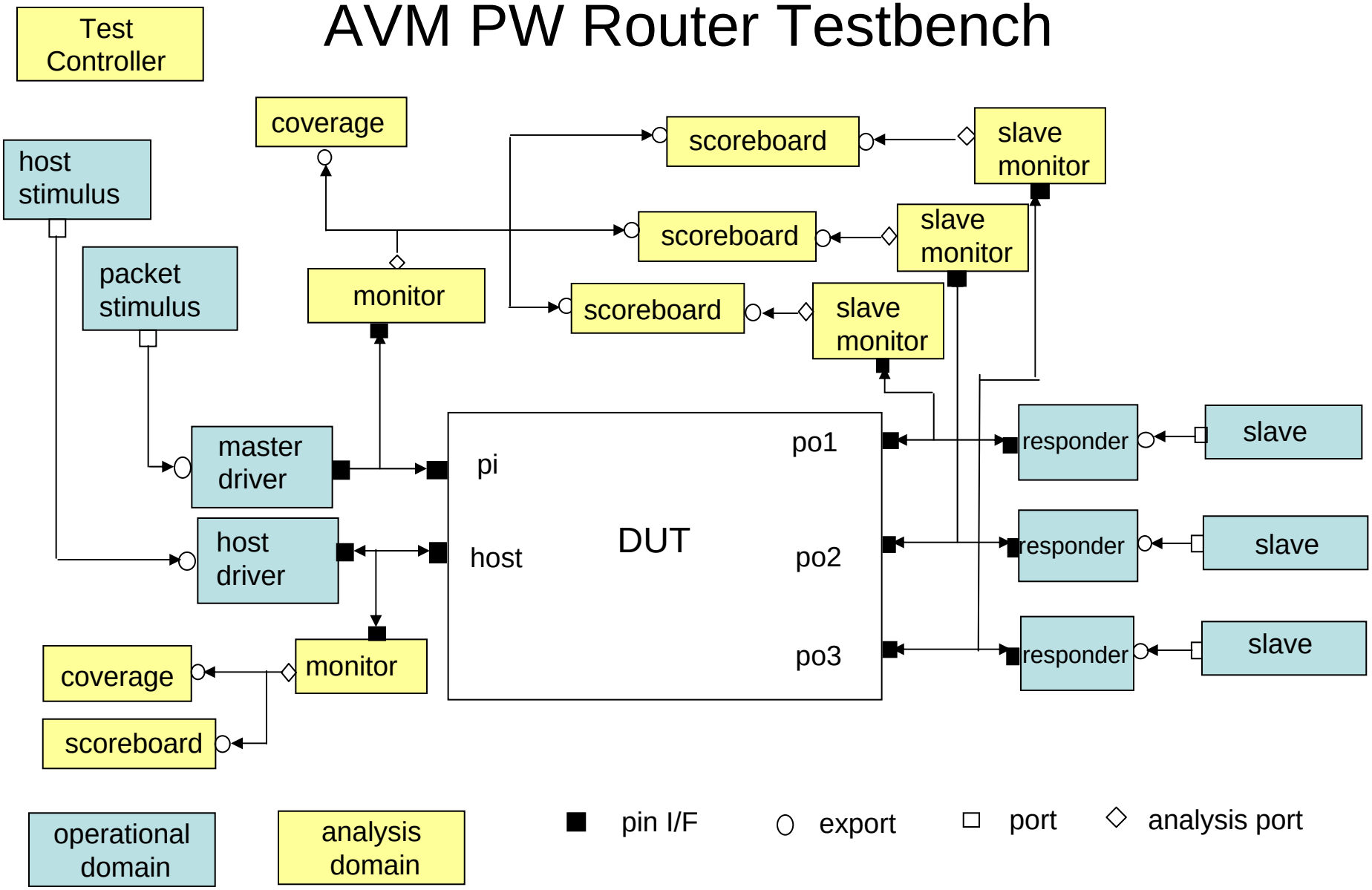
- **Testbench Overview**
  - AVM & URM
  - Compare AVM & URM
  - OVM testbench overview
- OVM Methodologies
  - Testbench configuration
  - Virtual Sequence
  - Error Test
- Conclusion

Design Under Test (DUT)

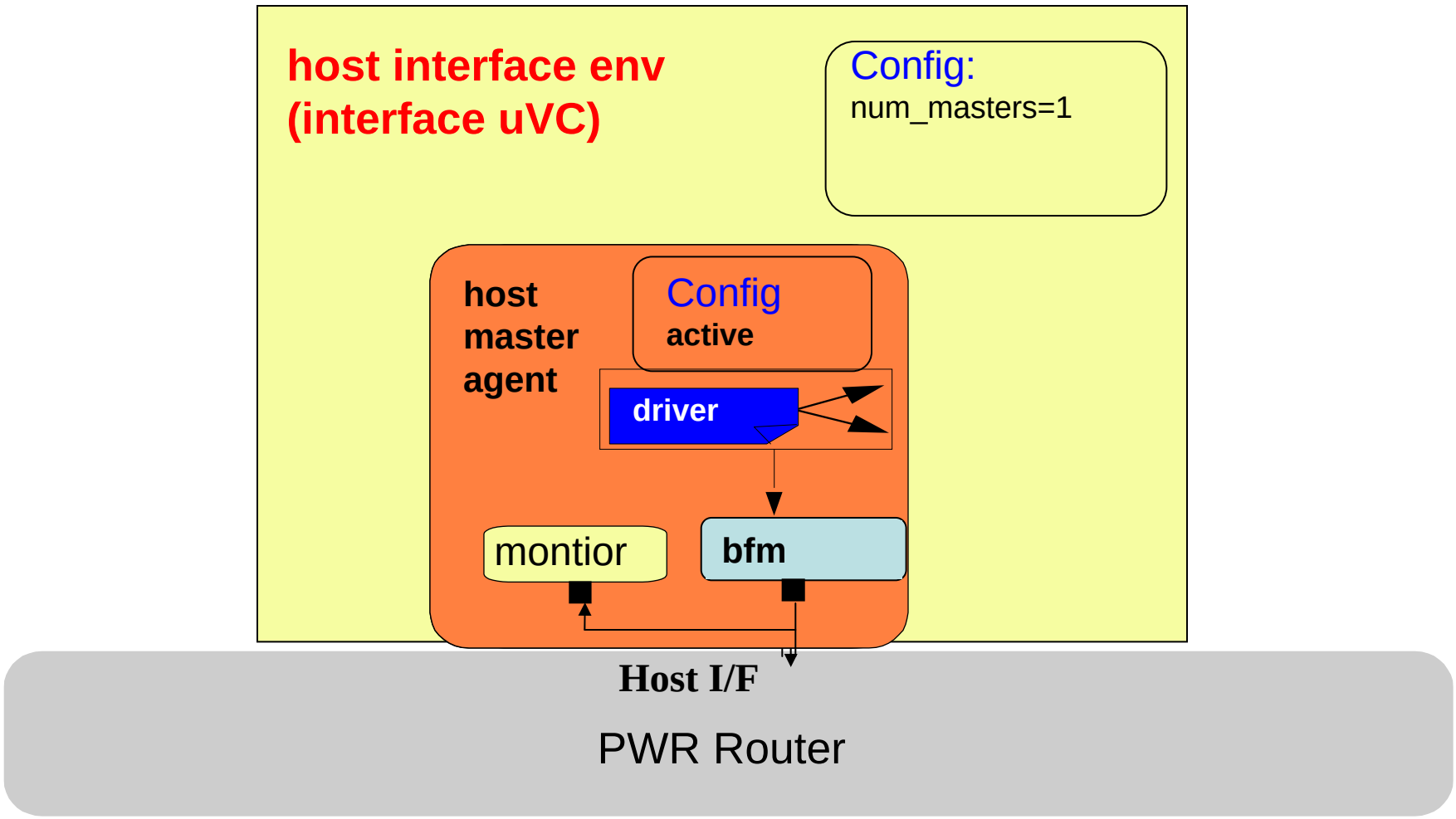
“PW Router”



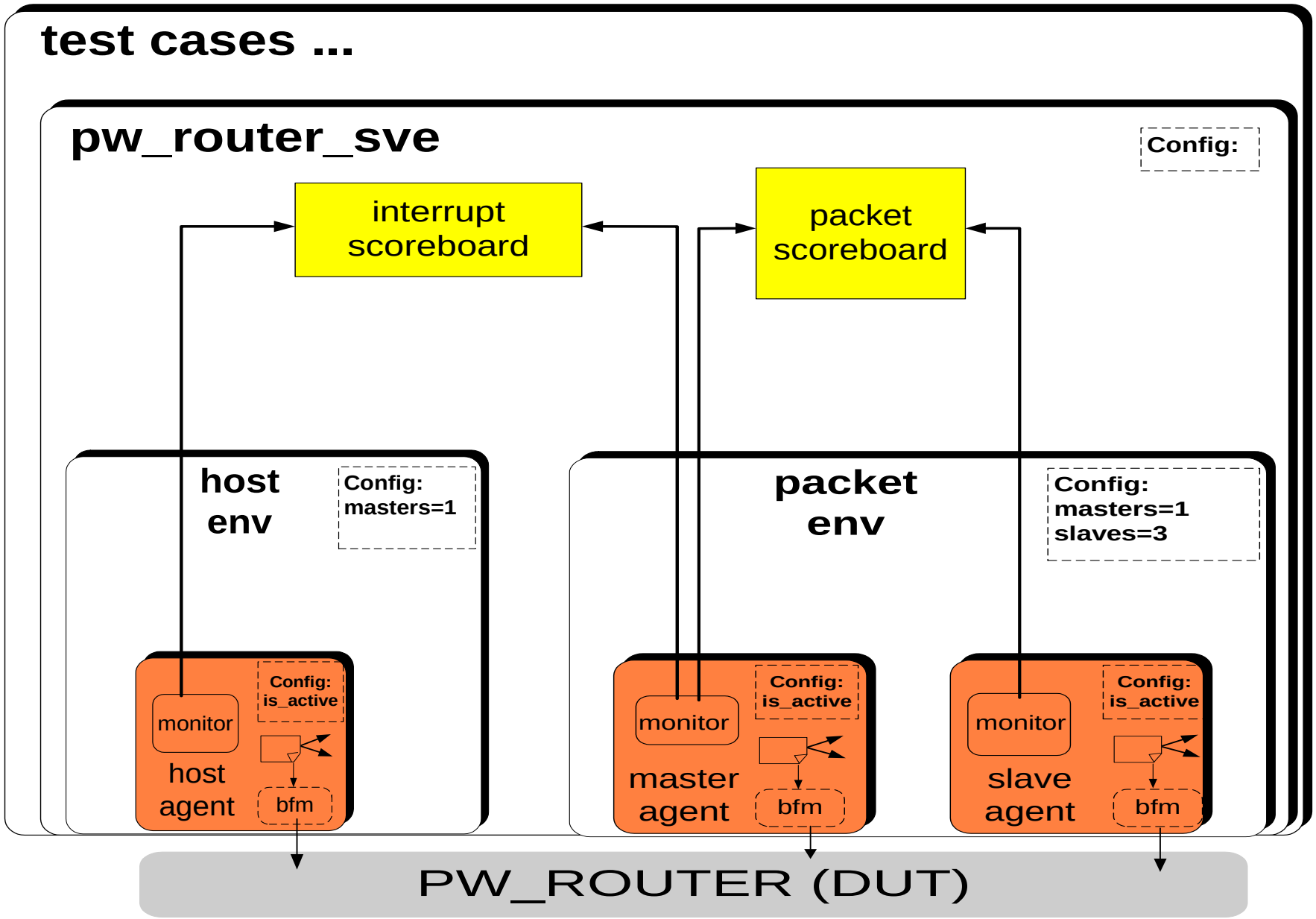
AVM PW Router Testbench



uRM Agent and Interface uVC



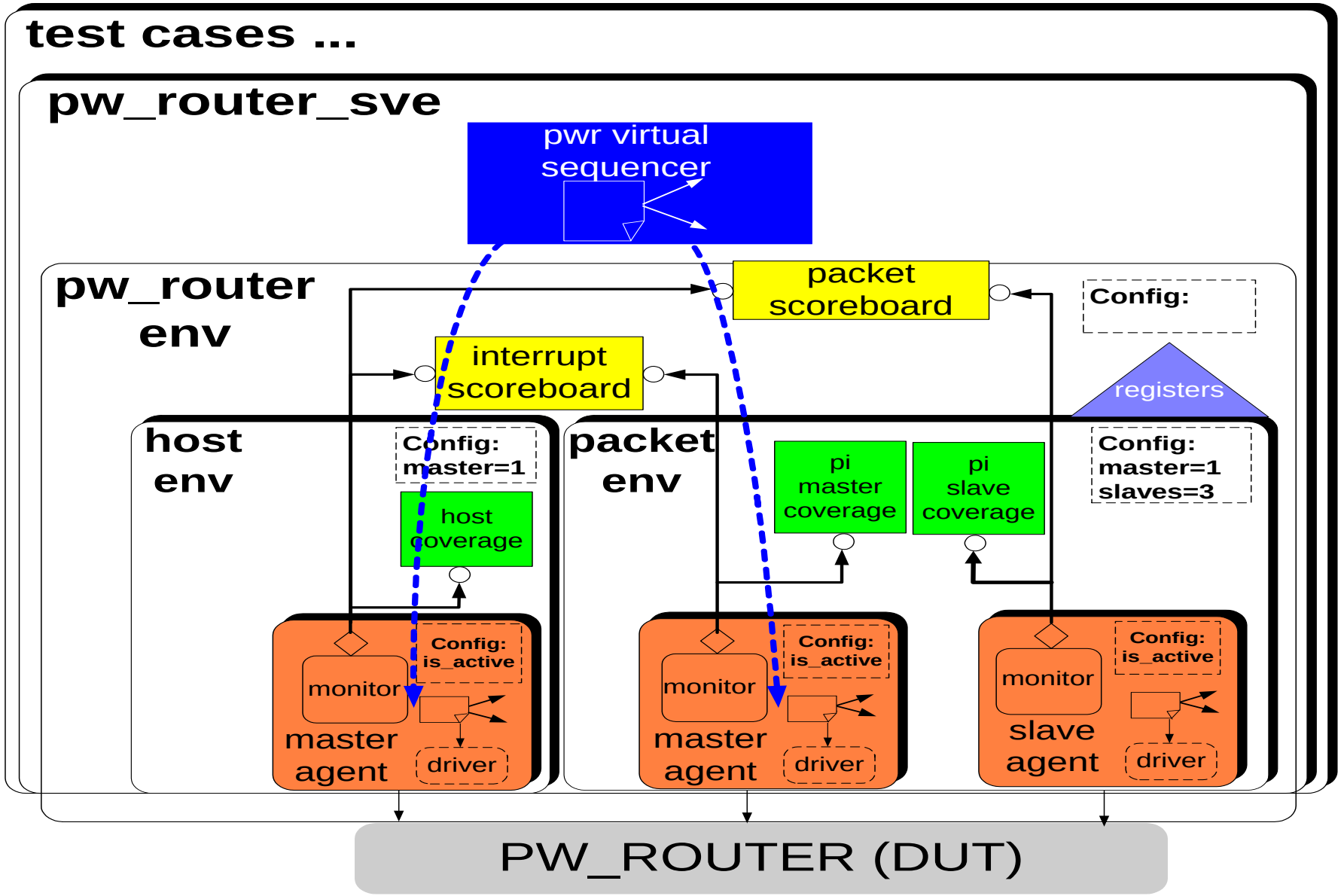
URM PW Router Testbench



AVM/URM Features

	AVM	URM	OVM
Testbench	- Test Controller - Operational Domain - Analysis Domain	Highly Layered -Test cases -SVE -uVCs	**Support both**
Components	TLM Channels/ Flat	Layered/ Structured / Configurable	**Support both**
Stimulus	avm_random_stimulus	Sequences	**Support both**
Configuration/ Factory	No AVM library features	Configuration/ Factory	**Configuration/ Factory**

OVM PW Router Testbench



URM to OVM Migration

```
class host_agent extends urm_agent;

  protected bit is_active = 1;
  pwr_parity_kind parity_kind;

  `urm_unit_utils_begin(host_agent)
    `urm_field_int(is_active, OVM_ALL_ON)
    `ovm_field_enum(pwr_parity_kind, parity_kind, OVM_ALL_ON)
`urm_unit_utils_end


  // new - constructor
  // new (string name, urm_object parent); does not work -
      prototype mismatch
  extern function new (string name, ovm_component parent);

endclass : host_agent
```

Able to reuse urm* componets!

-Able to use most MACRO utilities!
- However, the utility fields need modification.

OVM adds enumerator field utility.

Constructors need modification.

Outline

- Testbench Overview
  - AVM & URM
  - Compare AVM & URM
  - OVM testbench overview
- **OVM Methodologies**
  - **Testbench configuration**
  - Virtual Sequence
  - Error Test
- Conclusion

Testbench Configuration

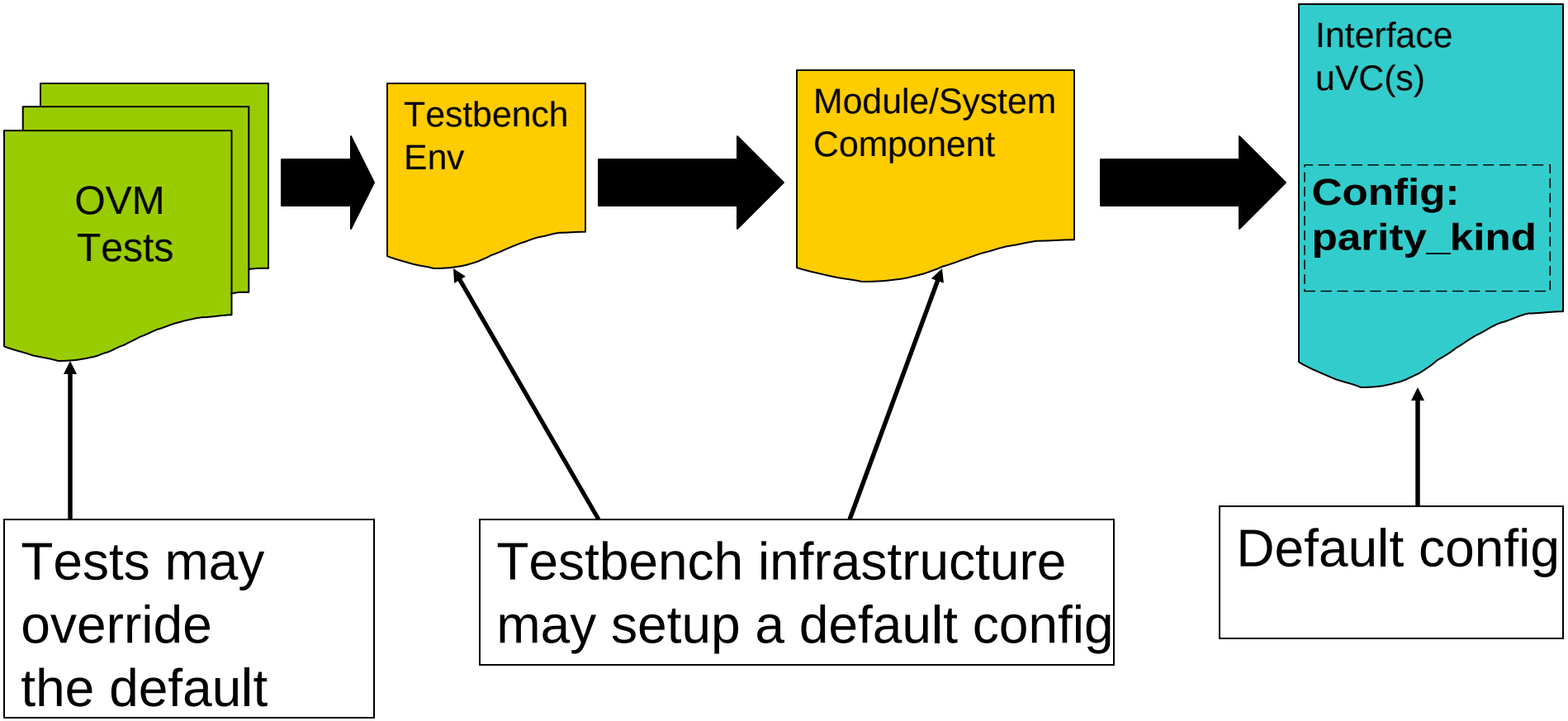
What needs to be configurable?

```
// Hierarchical Fields : Example 1
class packet_env extends ovm_env;
 int unsigned num_masters; // number of master agent
 int unsigned num_slaves;    // number of slave agent
```

```
// Hierarchical Fields : Example 2
class packet_agent extends ovm_agent;
 ovm_active_passive_enum is_active; // {ACTIVE,PASSIVE}
```

```
// Behavioral Field
class host_sequencer extends ovm_agent;
 pwr_parity_kind   parity_kind;  // user defined enum {ODD,EVEN}
```

Testbench Configuration (con't)



Testbench Configuration (con't)

STEP 1 – Add field in uVCs and register it with the factory

```
class packet_sequencer extends ovm_seqeuncer;

pwr_parity_kind parity_kind;
```

```
class host_seqeuncer extends ovm_seqeuncer;

pwr_parity_kind parity_kind;

`ovm_component_utils_begin(host_sequencer)
    `ovm_field_enum (pwr_parity_kind, parity_kind, OVM_ALL_ON)
`ovm_component_utils_end
```

Register with factory

TYPE

VALUE

FIELD

Since **parity_kind** is used by **more** than 1 uVC we add it value in the env

```
class pwr_env extends ovm_env;

rand pwr_parity_kind parity_kind;

 ovm_component_utils_begin(pwr_env)
   `ovm_field_enum (pwr_parity_kind, parity_kind, OVM_ALL_ON)
`ovm_component_utils_end
...
```

By default parity_kind is **random (ODD or EVEN)**!

Testbench Configuration (con't)

STEP 2 – Randomize the pwr's env in the sve

```
class pwr_sve_env extends ovm_env;

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

   $cast(pwr0, create_component("pwr_env", "pwr0"));
    assert(pwr0.randomize());
   …

 endfunction : build
```

Randomize after creating pwr0 component and building pwr0

Testbench Configuration (con't)

STEP 3 – Push down env's parity_kind to the uVCs

```
class pwr_env extends ovm_env;

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

    set_config_int(      "*"     , "parity_kind", parity_kind );

      //  host env sub-component
    $cast(host0, create_component("host_env", "host0"));

     // pi env sub-component
     $cast(pi0, create_component("packet_env", "pi0"));
```

path

field

value

set_config*
- Searches top-down
-May use wildcards

Note: make sure to call set_config* **after** super.build

Hint: for debugging set_config_* call host0.print() and pi0.print() here

Testbench Configuration (con't)

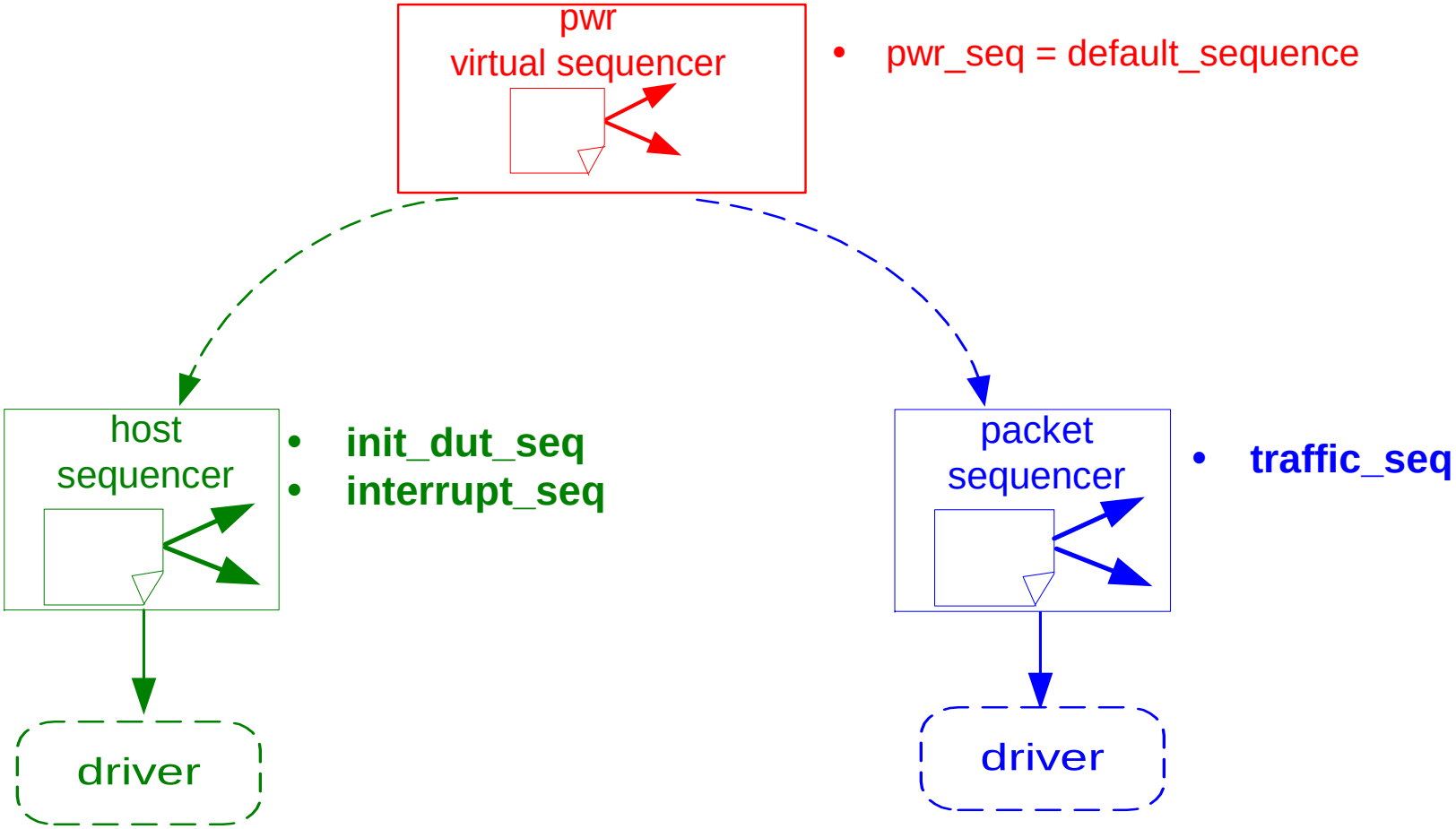
STEP 4 – Tests may override the env's default configuration

```
class test_odd_parity_kind extends pwr_base_test;

  `ovm_component_utils(test_odd_parity_kind)

virtual function void build();

    set_config_int("pwr_sve0.pwr0", "parity_kind", ODD);

   super.build();
  endfunction : build

endclass : test_odd_packet_parity
```

Outline

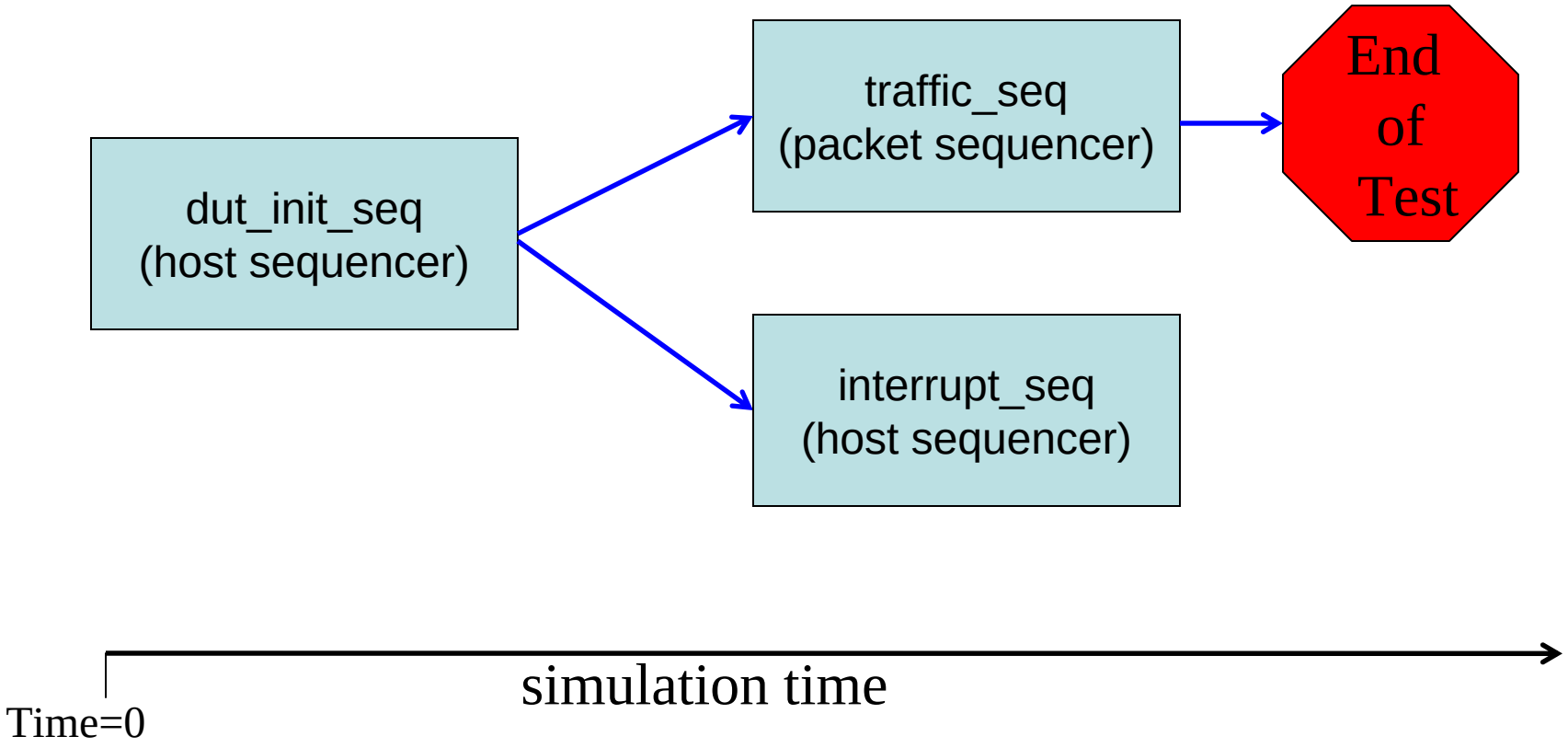
- Testbench Overview
  - AVM & URM
  - Compare AVM & URM
  - OVM testbench overview
- **OVM Methodologies**
  - Testbench configuration
  - **Virtual Sequence**
  - Error Test
- Conclusion

Virtual Sequencer



PWR Virtual Sequence

pwr_seq virtual sequence progression



PWR Virtual Sequence (con't)

```
class pwr_seq extends ovm_sequence;
`ovm_sequence_utils(pwr_seq, pwr_virtual_sequencer)

  init_dut_seq      init_dut_seq_inst;  // host sequencer
  interrupt_seq     interrupt_seq_inst; // host sequencer
  traffic_seq       traffic_seq_inst;     // packet sequencer

virtual task body();

    `ovm_do_seq(init_dut_seq_inst, p_sequencer.seq_cons_if["host_sequencer"])

    fork
      begin
        `ovm_do_seq(traffic_seq_inst,p_sequencer.seq_cons_if["packet_sequencer"])
      end
      begin
        `ovm_do_seq(interrupt_seq_inst, p_sequencer.seq_cons_if["host_sequencer"])
      end
    join_any
  endtask
```

Register sequence

Instantiate sequences

Call Init DUT with Host Sequencer

Sequence instance

Sequencer

Call traffic with Packet Sequencer

Call Interrupt with Host Sequencer

Outline

- Testbench Overview
  - AVM & URM
  - Compare AVM & URM
  - OVM testbench overview
- **OVM Methodologies**
  - Testbench configuration
  - Virtual Sequence
  - **Error Test**
- Conclusion

Simple Error Test

STEP 1 – Create new inherited traffic sequence

```
class my_error_traffic_seq extends traffic_seq;

  `ovm_sequence_utils(my_error_traffic_seq, packet_sequencer)

    pi_transfer this_transfer;

    virtual task body();
      `ovm_create(this_transfer)     // create a variable for manipulation
      this_transfer.parity_error_c.constraint_mode(0); // turn off default constraint
      `ovm_rand_send_with(this_transfer, { parity_error == 1'b1; } )
     endtask

endclass
```

Register sequence

Force a parity error!

Simple Error Test (con't)

STEP 2 – Create test that calls the error traffic sequence

```
class test_error_packet extends pwr_base_test;

  `ovm_component_utils(test_error_packet)

 ...

virtual function void build();
 ovm_factory::set_type_override( "traffic_seq",
                                 "my_error_traffic_seq");
 ----- OR -----
 ovm_factory::set_inst_override( "*traffic_seq_inst",
                                 "traffic_seq ",
                                 "my_error_traffic_seq" );
  // Create the sve
    super.build();
  endfunction : build
endclass
```

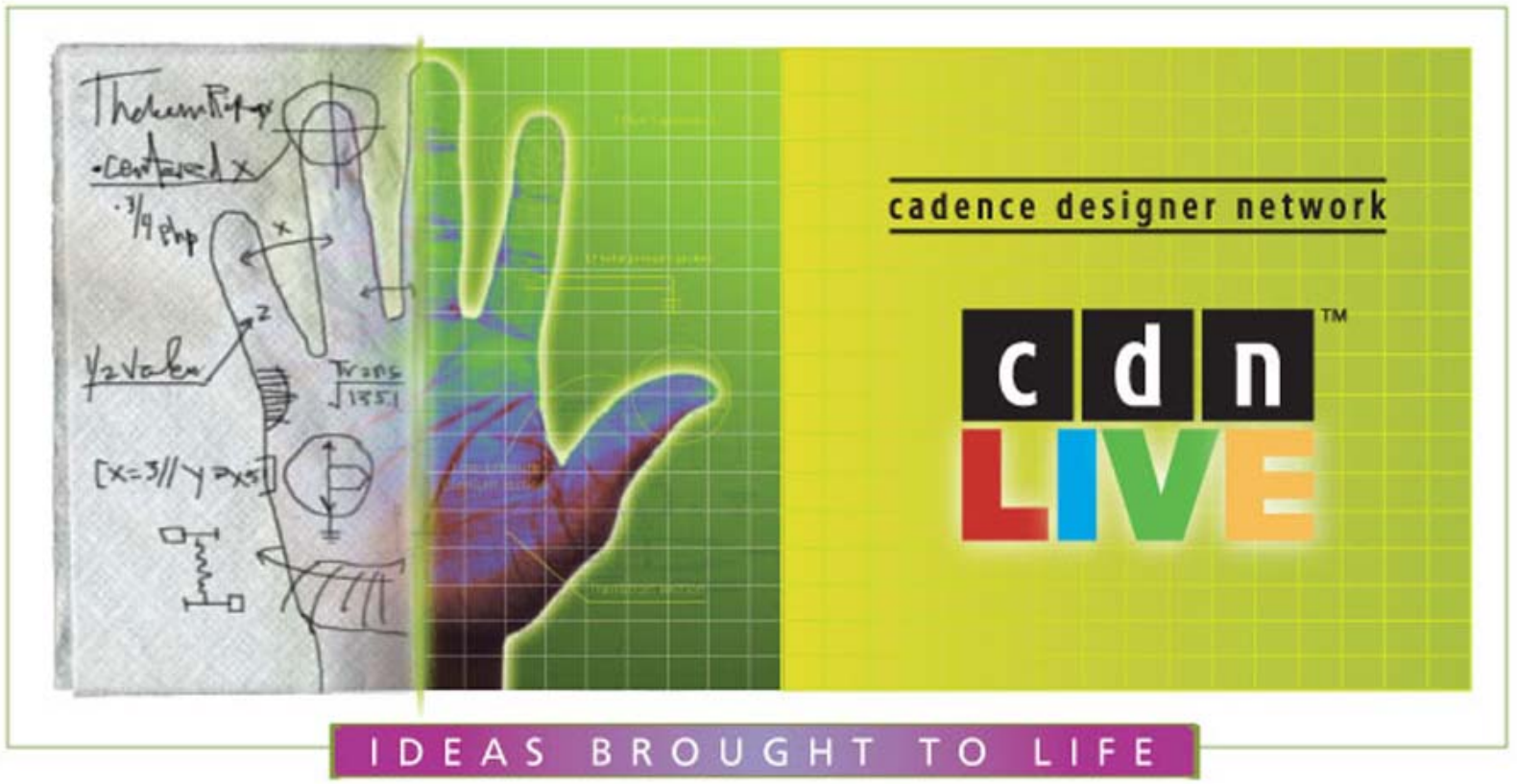
These factory overrides will force "my_error_traffic_sequence" to be invoked!

Hint: for debugging purposes call out ovm_factory::print_all_overrides();

OVM Conclusion

- Migration effort was minimal
- Works and proven based on eRM/AVM
- Upfront learning curve
- Promotes readable and maintainable code
- Powerful for reuse
  - Virtual Sequences
  - Configuration Management & Factory
  - Hierarchal Structure
  - TLM

cadence



Contact Information:

stephen.donofrio@paradigm-works.com
www.paradigm-works.com