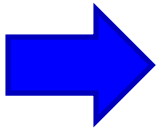
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Doing Funny Stuff with the UVM Register Layer: Experiences Using Front Door Sequences, Predictors, and Callbacks

John Aynsley, Doulos

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda



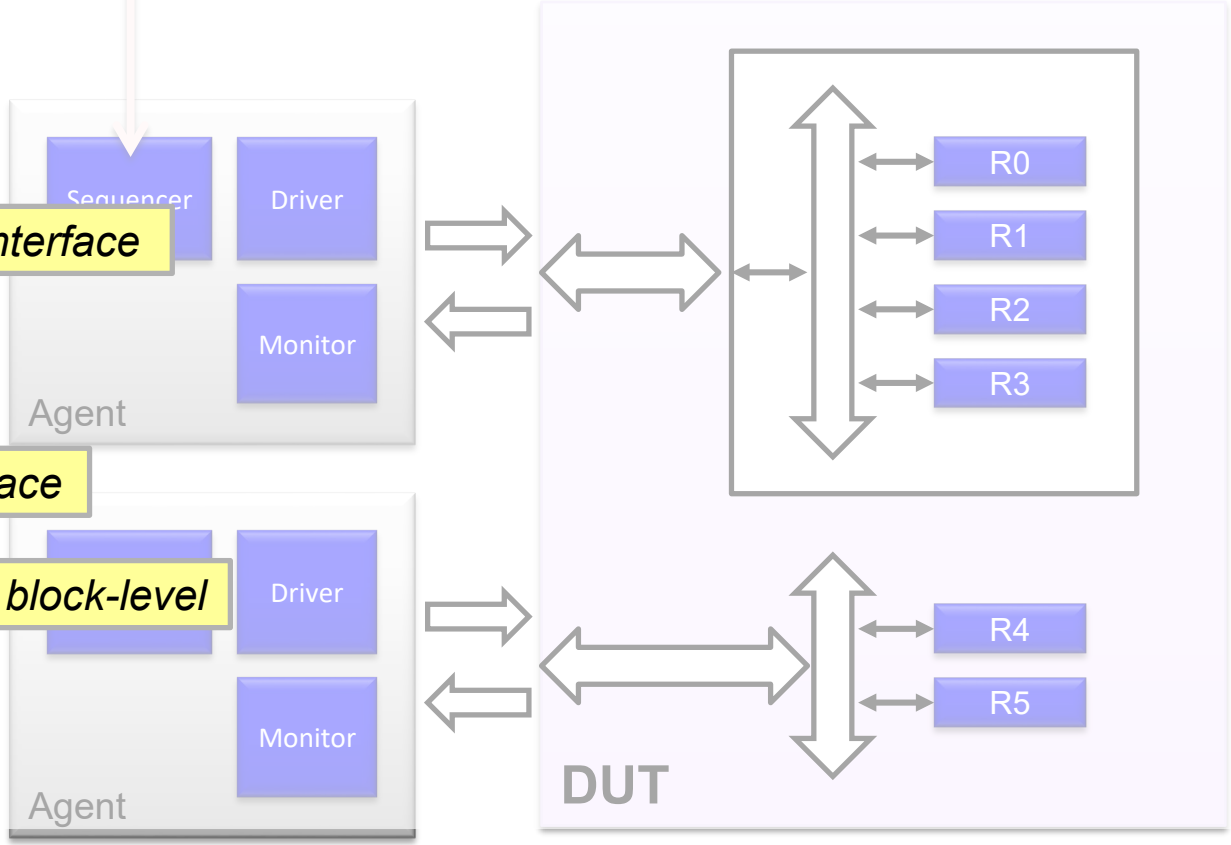
- Introduction
- User-defined front door sequences
- User-defined back doors
- The predictor
- Register callbacks

Why the Register Layer?

```
tx.randomize() with { cmd == c; addr == a; data == d; };
```

Test dependent on:

- *choice of physical interface*
- *physical address*
- *endianness of interface*
- *system-level versus block-level*



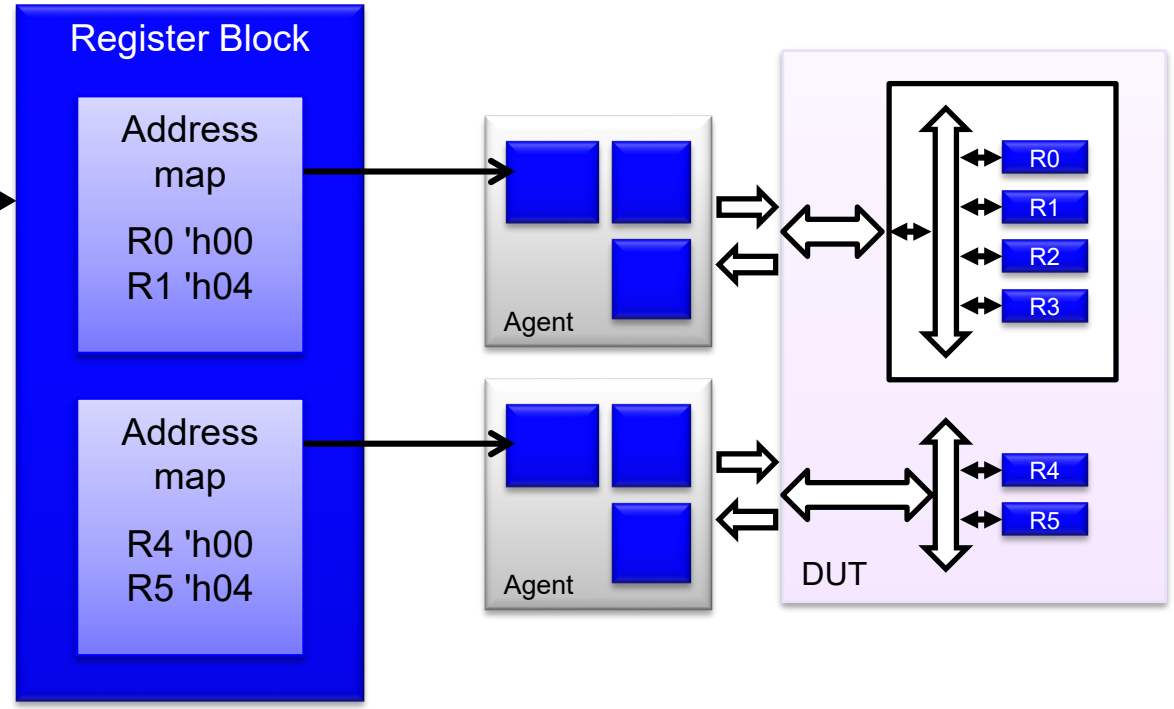
The UVM Register Layer

From test

```
R0.write(value);
R1.read(value);
```

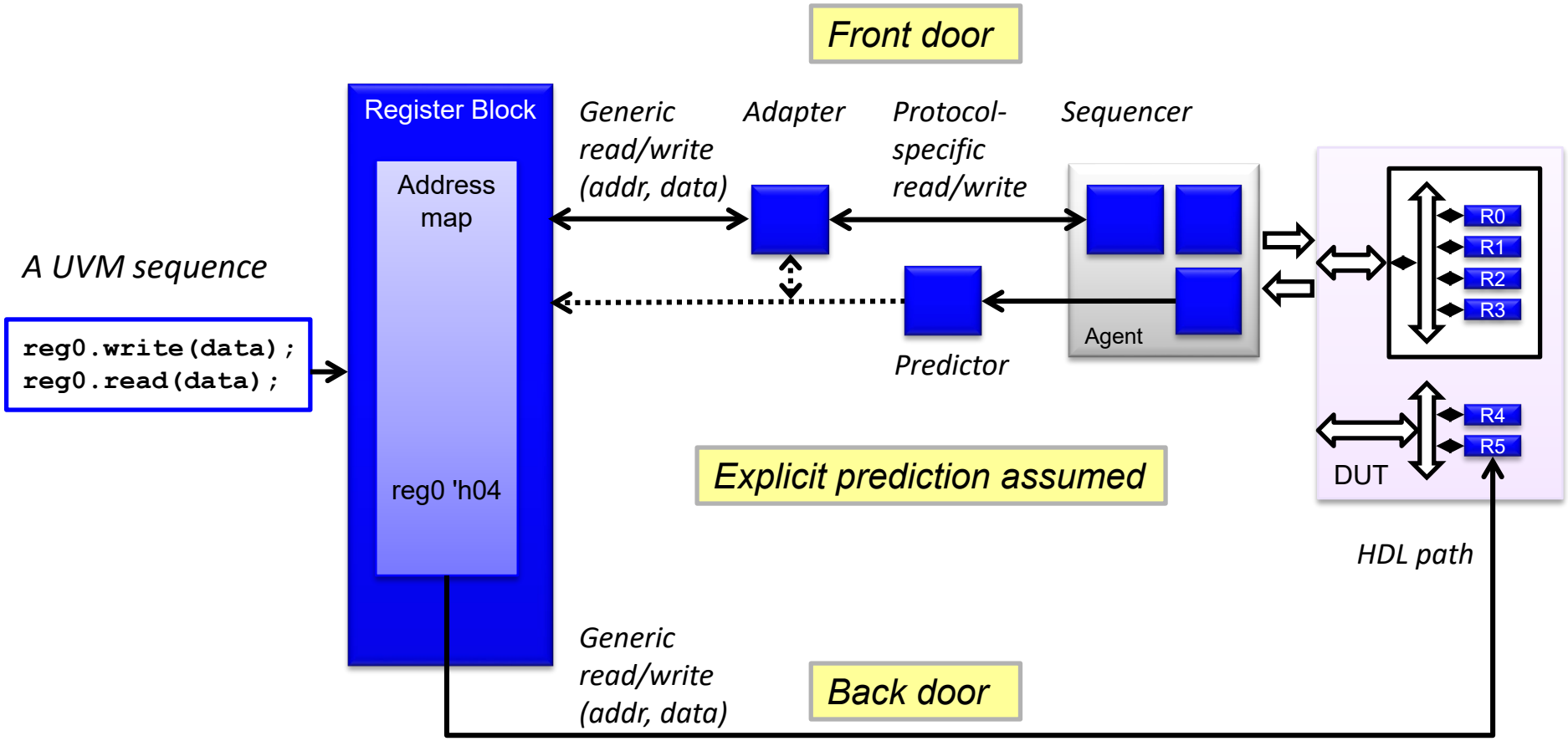
No protocol
No address
No endianness

Anywhere in DUT



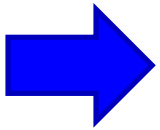
Need a generator!

Integration with the DUT



2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

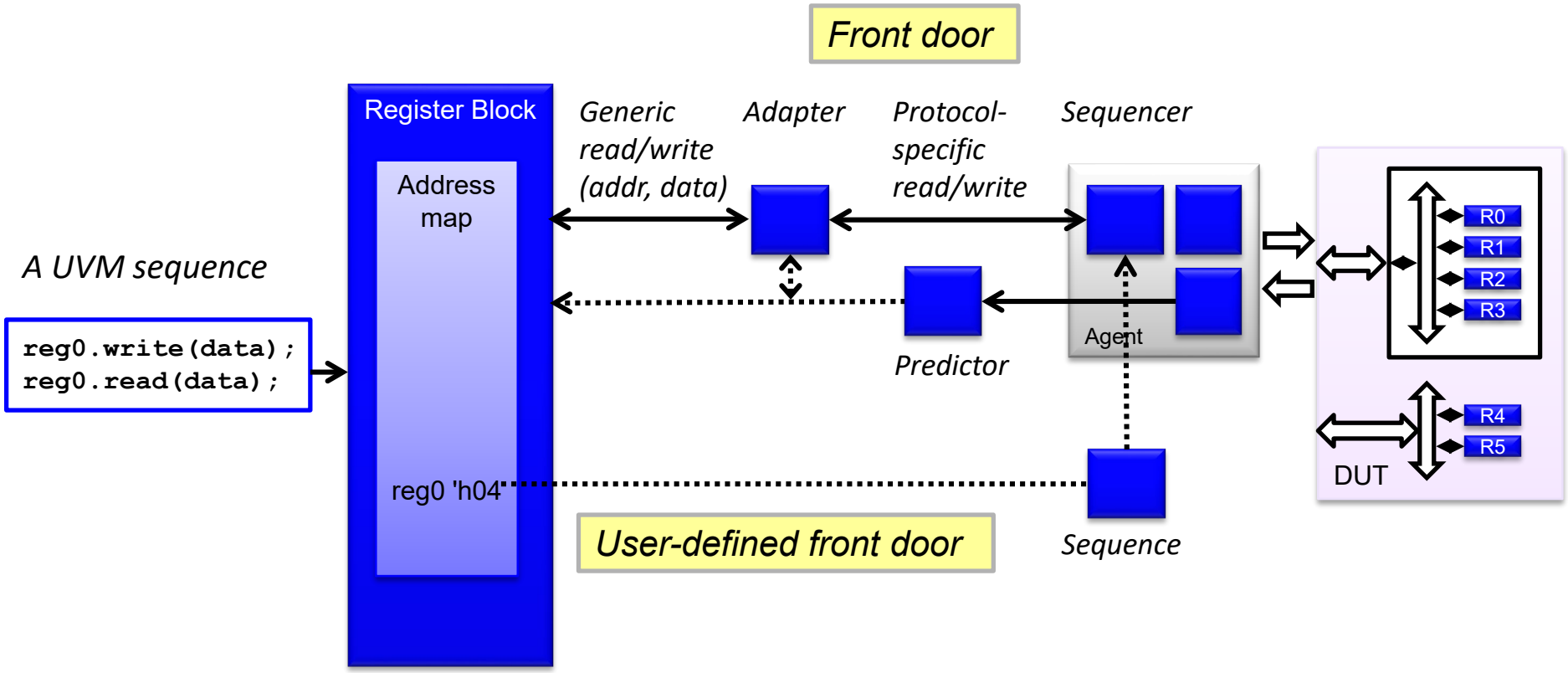


- Introduction
- User-defined front door sequences
- User-defined back doors
- The predictor
- Register callbacks

Funny Stuff

- Less obvious ways to address a register
  - Non-linear addressing
  - Burst access mode
  - Indirect access though an embedded CPU
  - Non-memory-mapped registers

User-Defined Front Door



Setting the Front Door

```
my_vreg_frontdoor_seq frontdoor;

frontdoor = my_vreg_frontdoor_seq::type_id::create("frontdoor");

regmodel.bus.reg0.set_frontdoor(frontdoor);

regmodel.bus.reg1.set_frontdoor(frontdoor);
```

Front Door Sequence 1

```
class my_vreg_frontdoor_seq extends uvm_reg_frontdoor;
  ...
  task body;
    uvm_reg          the_reg;
    uvm_reg_addr_t   reg_addr;
    bit              cmd;
    uvm_reg_data_t   data;

    $cast(the_reg, rw_info.element);

    reg_addr = the_reg.get_offset();

    cmd  = (rw_info.kind == UVM_WRITE);
    ...
```

Find the original uvm_reg object

Find the address of the register

Front Door Sequence 2

```
  ...
  data = rw_info.value[0][3:0];

  one_transaction( .cmd(cmd), .addr(reg_addr+1), .data(data) );

  if (cmd == 0)
    rw_info.value[0][3:0] = data;

  data = rw_info.value[0][7:4];

  one_transaction( .cmd(cmd), .addr(reg_addr+5), .data(data) );

  if (cmd == 0)
    rw_info.value[0][7:4] = data;

  rw_info.status = UVM_IS_OK;
endtask
```

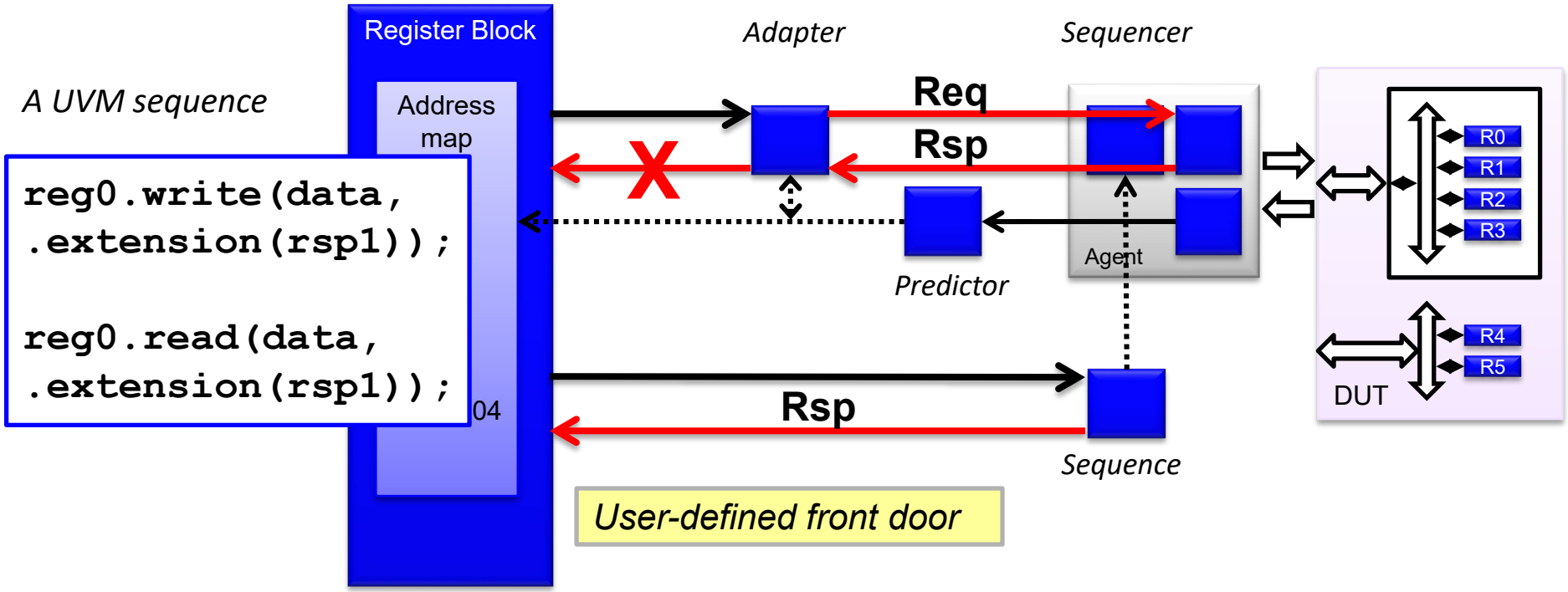
Bottom nibble

Read command

Top nibble

Read command

Passing Back a Response



Response object created first and passed to write/read

Rsp from Front Door Sequence

```
task body;
  ...
  finish_item(req);

  get_response(item);

  $cast(rsp, item);
  assert(rsp != null);

  rw_info.extension.copy(rsp);

  if (cmd == 0)
    rw_info.value[0] = rsp.data;
  ...
```

Get response object from driver

Copy rsp into the extension argument

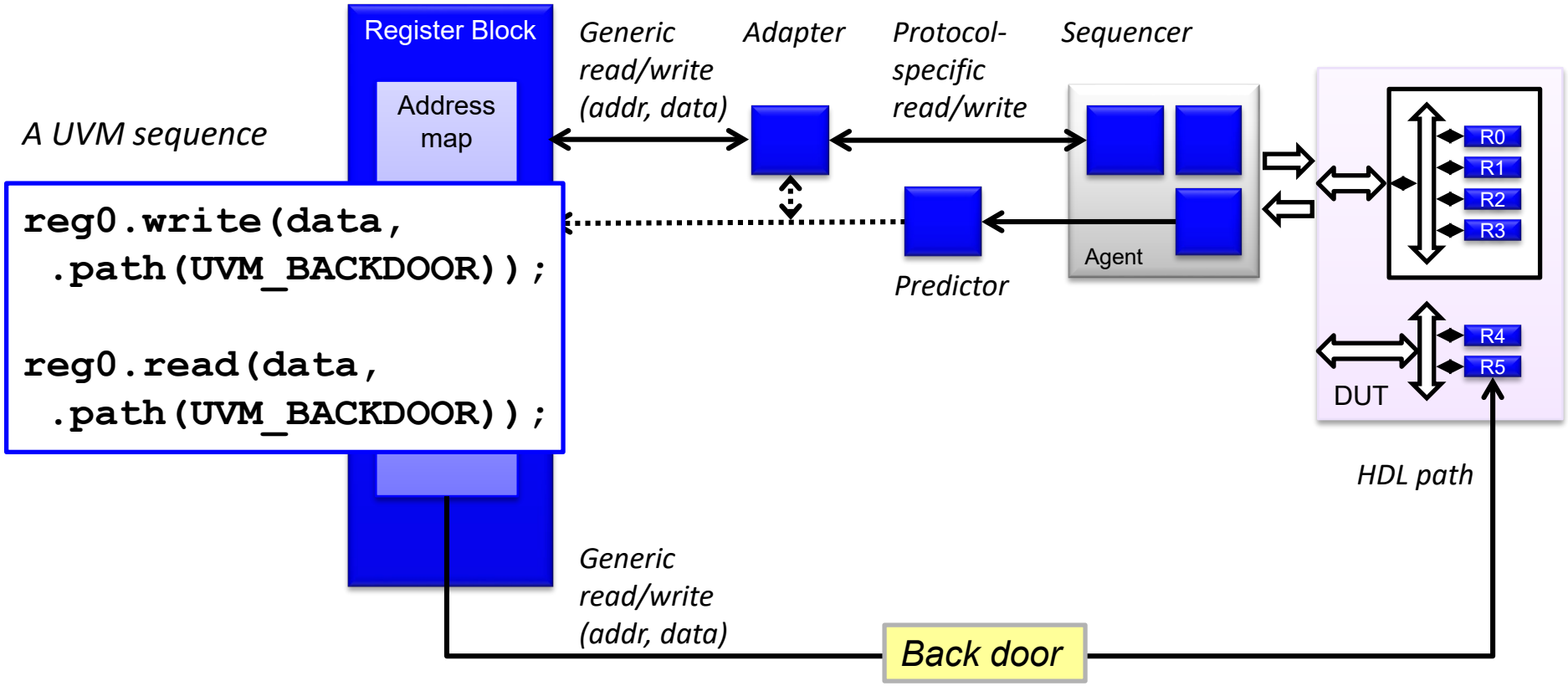
Copy data into the value argument

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- User-defined front door sequences
- User-defined back doors
- The predictor
- Register callbacks

Using the Back Door



A User-Defined Back Door

```
my_mem_backdoor backdoor;

backdoor = my_mem_backdoor::type_id::create("backdoor");

regmodel.bus.mem.set_backdoor(backdoor);
```

Memories generally use back door access

User-Defined Back Door 1

```
class my_mem_backdoor extends uvm_reg_backdoor;
  `uvm_object_utils(my_mem_backdoor)

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

  virtual task write(uvm_reg_item rw);
    bit ok;
    int n = rw.value.size();
    for (int i = 0; i < n; i++)
    begin
      ok = uvm_hdl_deposit(
           $sformatf("top_tb.th.uut.mem[%0d]", rw.offset + i),
           rw.value[i]);
      assert(ok);
    end
    rw.status = UVM_IS_OK;
  endtask
  ...
```

Not a sequence

rw versus rw_info

User-Defined Back Door 2

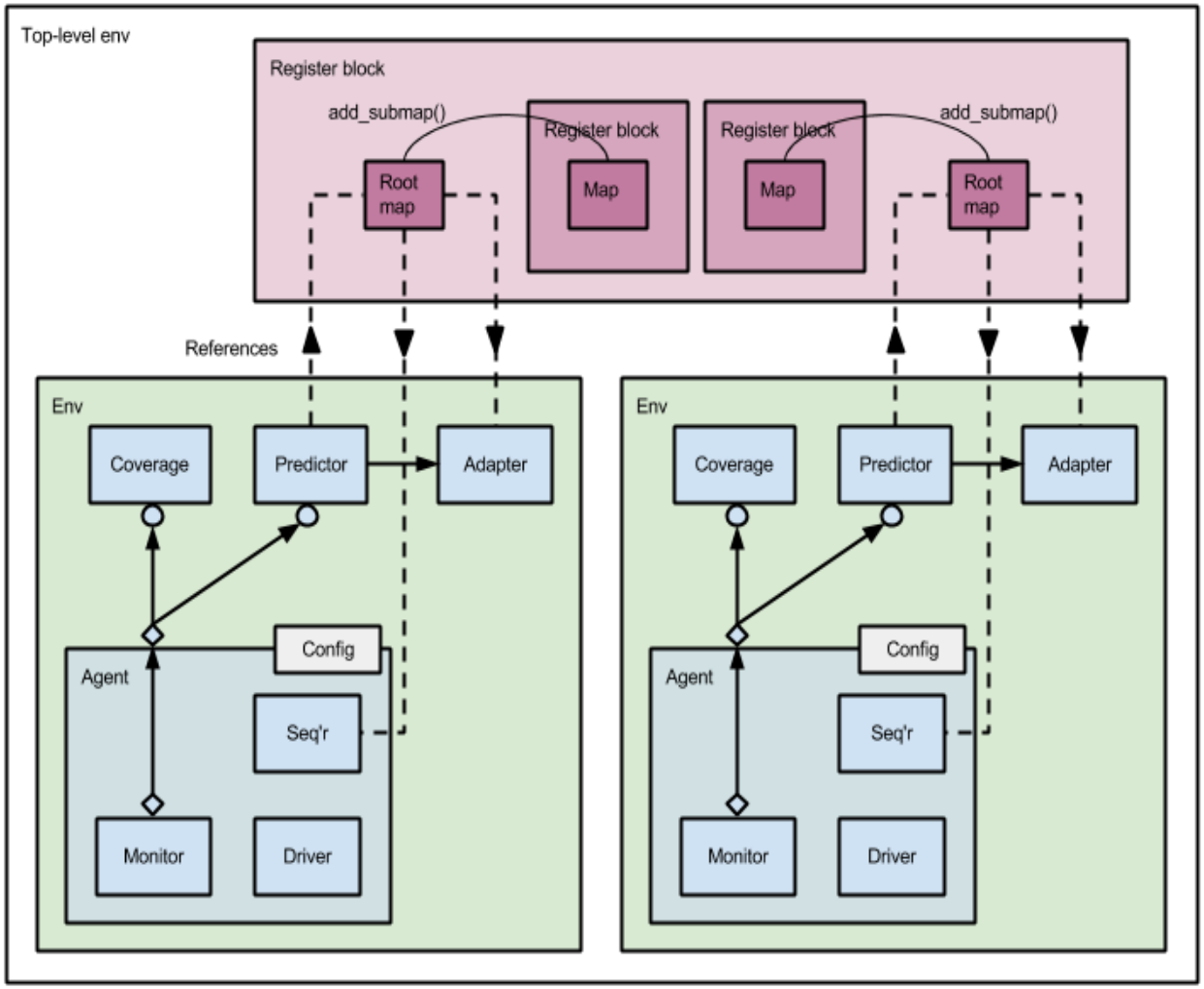
```
  ...
  virtual task read(uvm_reg_item rw);
    bit ok;
    int n = rw.value.size();
    for (int i = 0; i < n; i++)
    begin
      ok = uvm_hdl_read(
           $sformatf("top_tb.th.uut.mem[%0d]", rw.offset + i),
           rw.value[i]);
      assert(ok);
    end
    rw.status = UVM_IS_OK;
  endtask

endclass
```

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- User-defined front door sequences
- User-defined back doors
- The predictor
- Register callbacks

Connecting the Predictor



Mirrored Value

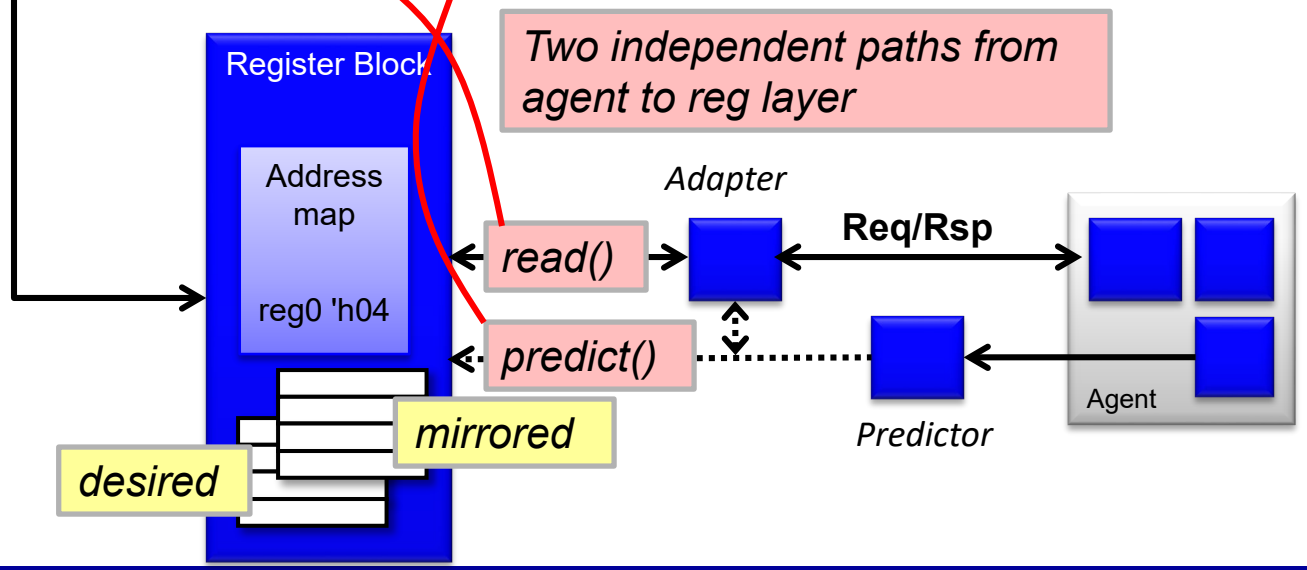
```
value = regmodel.reg0.get_mirrored_value();

regmodel.reg0.set( .value('hab) );

regmodel.reg0.update( .status(status) );

regmodel.update( .status(status) );

regmodel.reg0.read( .value(data) );
```

Return mirror without accessing DUT

Set the desired value

Write desired value to register

Only write if desired != mirrored



2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- User-defined front door sequences
- User-defined back doors
- The predictor
- Register callbacks

Funny Stuff

- Quirky registers
- Side-effects of register reads and writes
- Aliased registers (see paper)

Built-in Access Policies

```
class fancy_reg extends uvm_reg;
   `uvm_object_utils(fancy_reg)

   rand uvm_reg_field F1;

   function new(string name = "");
      super.new(name, 16, UVM_NO_COVERAGE);
   endfunction

   virtual function void build();
      F1 = uvm_reg_field::type_id::create("F1");
      F1.configure(this, 16, 0, "WSRC", 1, 16'h0000, 1, 1, 0);
   endfunction
endclass
```

Mirrored value will be set ('hffff) on write, cleared ('h0000) on read

Add Callbacks to Register

pre_read()
pre_write()
post_read()
post_write()
post_predict()

Add to register or to register field

```
my_reg_callbacks cb;

cb = new;

uvm_reg_cb::add(regmodel.bus.reg0, cb);
```

```
uvm_reg_field_cb::add(regmodel.bus.reg0.F1, cb);
```

Register Callbacks

```
class my_reg_callbacks extends uvm_reg_cbs;

  task post_read(uvm_reg_item rw);
    bit ok;
    uvm_reg the_reg;
    assert(rw.element_kind == UVM_REG);
    $cast(the_reg, rw.element);
    ok = the_reg.predict(0, -1, UVM_PREDICT_DIRECT);
    assert(ok);
  endtask

  task post_write(uvm_reg_item rw);
    bit ok;
    uvm_reg the_reg;
    assert(rw.element_kind == UVM_REG);
    $cast(the_reg, rw.element);
    ok = the_reg.predict(~rw.value[0], -1, UVM_PREDICT_DIRECT);
    assert(ok); ...
```

Only registers, not fields

predict() modifies the mirrored value

or UVM_PREDICT_READ or UVM_PREDICT _WRITE

Pitfalls

- post_predict() only called for register fields and only when predictor receives a transaction (assuming explicit prediction)
- post_predict() must not call predict() of the same register
- Best to call predict() from pre/post_read/write()

Any Questions?

https://www.edaplayground.com/x/55Wh

UVM non-linear register addressing

https://www.edaplayground.com/x/58jv

UVM register - extension argument to read/write

https://www.edaplayground.com/x/6DJg

UVM register user-defined memory backdoor

https://www.edaplayground.com/x/4apk

UVM register with side-effects

https://www.edaplayground.com/x/3FUt

UVM register aliasing

or search published Playgrounds for funny stuff