

DAC 2010 OVM World Booth

**The Communication and
Customization Mechanisms
in OVM and UVM**

John Aynsley, CTO, Doulos



The Communication and Customization Mechanisms in OVM and UVM

CONTENTS

- *Introducing Doulos*





Delivering Know-How

www.doulos.com

Hardware Design

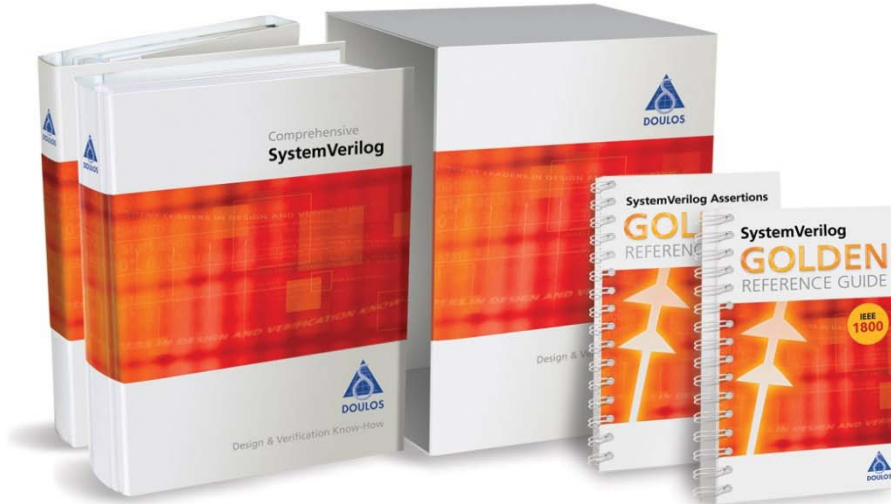
- » VHDL
- » Verilog
- » SystemVerilog
- » Actel
- » Altera
- » Xilinx

Embedded Systems and ARM

- » C
- » C++
- » UML
- » RTOS
- » Linux
- » ARM Cortex A/R/M series

ESL & Verification

- » SystemC
- » TLM-2.0
- » SystemVerilog
- » OVM/VMM/UVM
- » Perl
- » Tcl/Tk



- OVM and UVM Adopter Class
- SystemVerilog Training Customized for OVM / UVM
- Public open-enrolment or private on-site classes

The Communication and Customization Mechanisms in OVM and UVM

CONTENTS

- *Introducing Doulos*
- *Let's get Technical...*
 - *Communication Mechanisms*



- Based on the SystemC TLM-1 standard

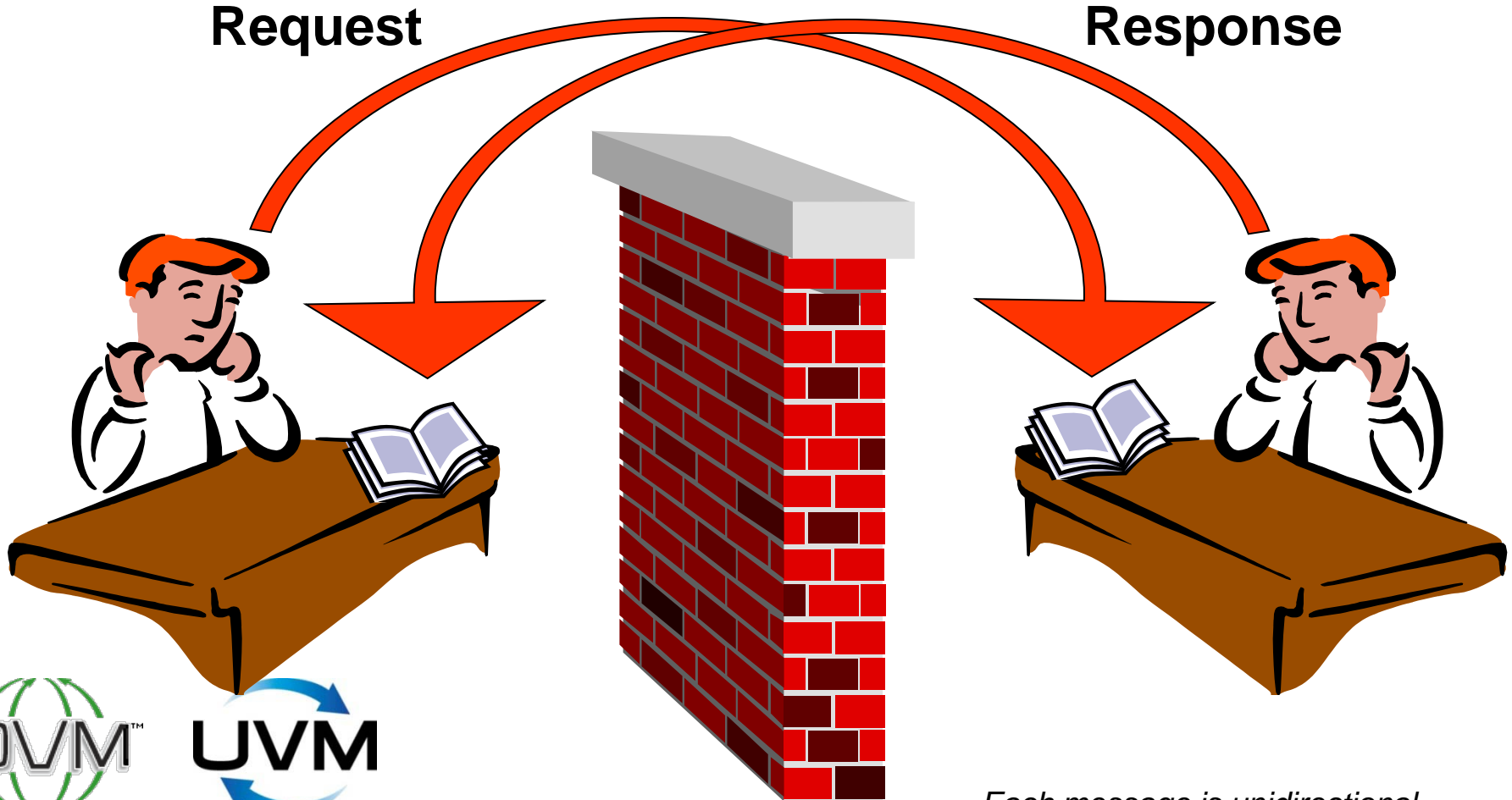
put

get

peek

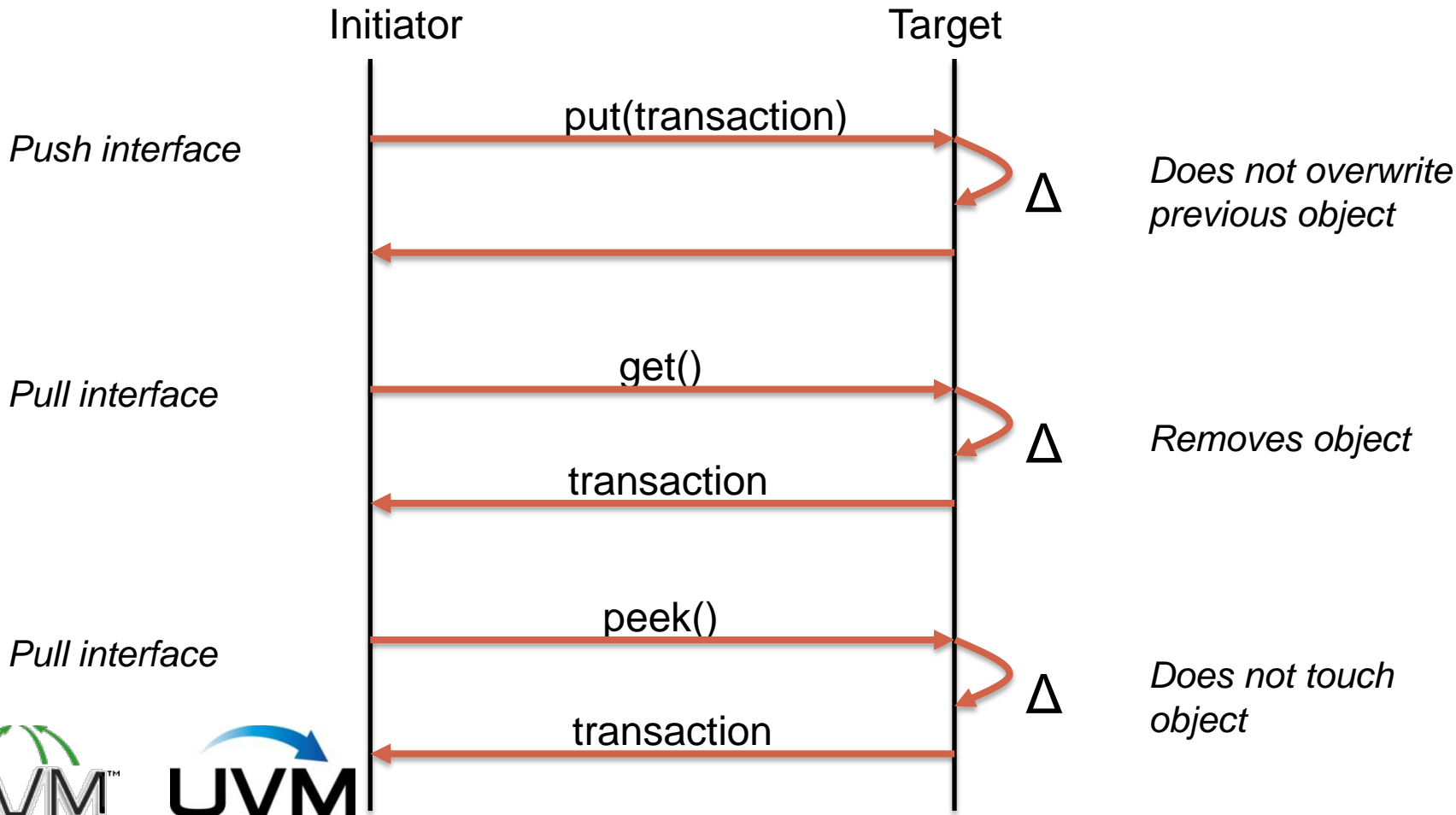


TLM-1 = Message Passing



Each message is unidirectional

TLM-1 Semantics



Blocking

put

get

peek

Non-blocking

try_put

try_get

try_peek

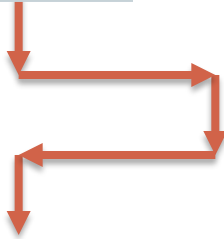
can_put

can_get

can_peek

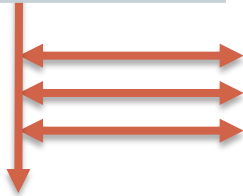
Blocking vs Non-blocking

put



- A blocking method
 - may suspend
 - only returns when it has succeeded

try_put



- A non-blocking method
 - must return immediately
 - returns a status flag (success/failure)
 - may require multiple tries

Required / Provided Interfaces



```
put_port.put(tx);
```

```
task put(T tx);
```

Required / Provided Interfaces



```
ovm_blocking_put port #(T) put_port;
```

Required

```
...  
put_port.put(tx);
```

```
ovm_blocking_put export #(T) put_export;
```

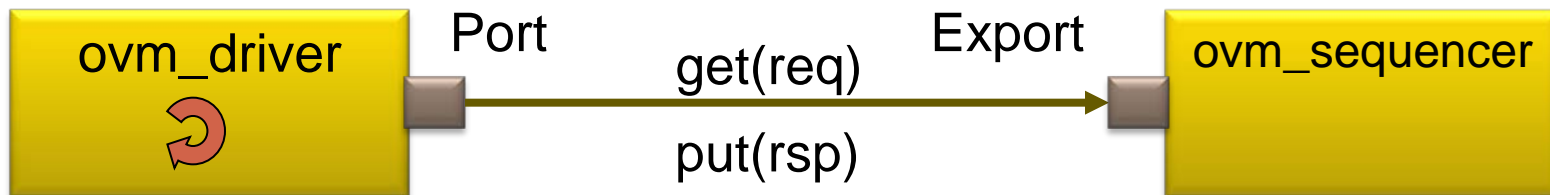
Provided

```
...  
task put(T tx);
```

Contract

```
prod.put_port.connect( cons.put_export );
```

Sequencer-Driver Communication



```
class my_driver extends ovm_driver #(REQ, RSP);
```

```
// ovm_seq_item_pull_port #(REQ, RSP) seq_item_port;
```

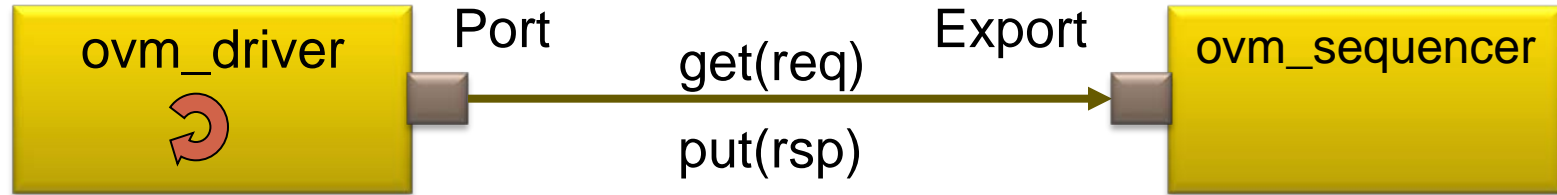
```
...
```

```
class my_sequencer extends ovm_sequencer #(REQ, RSP);
```

```
// ovm_seq_item_pull_imp #(REQ, RSP, this_type) seq_item_export;
```

```
...
```

Sequencer-Driver Communication



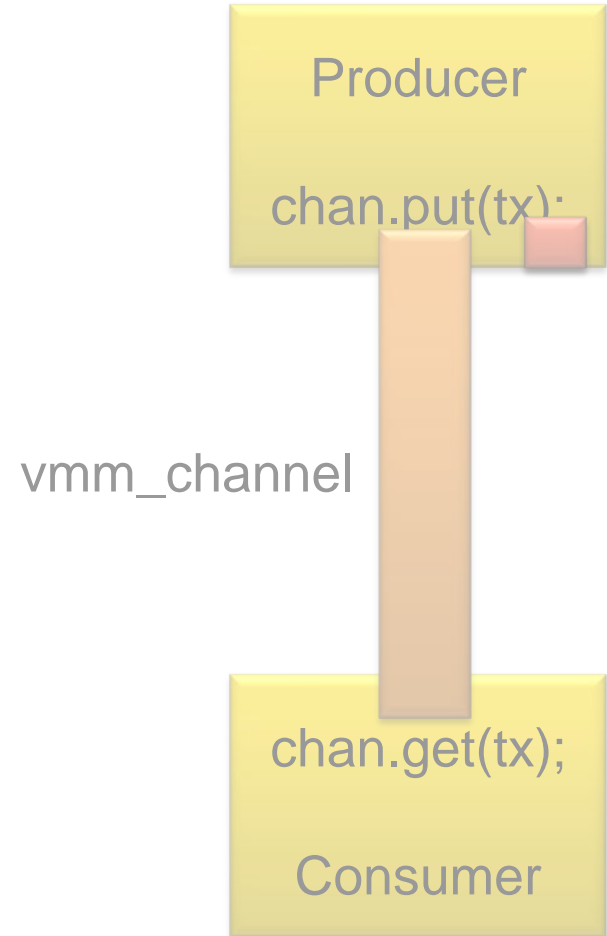
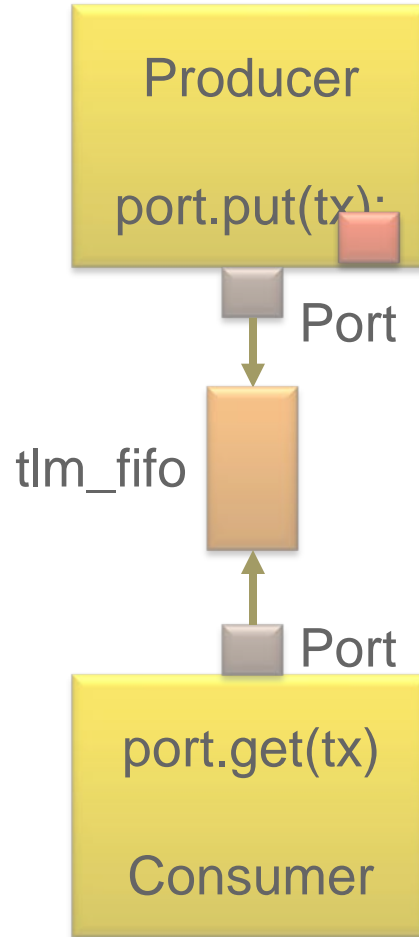
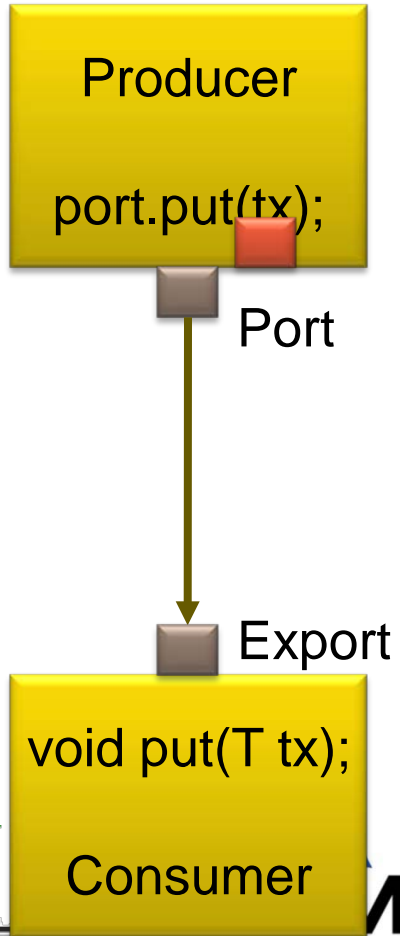
```
class my_agent extends ovm_agent;
...
my_sequencer sequencer_h;
my_driver      driver_h;

virtual function void connect;
    driver_h.seq_item_port.connect( sequencer_h.seq_item_export );
endfunction: connect
...

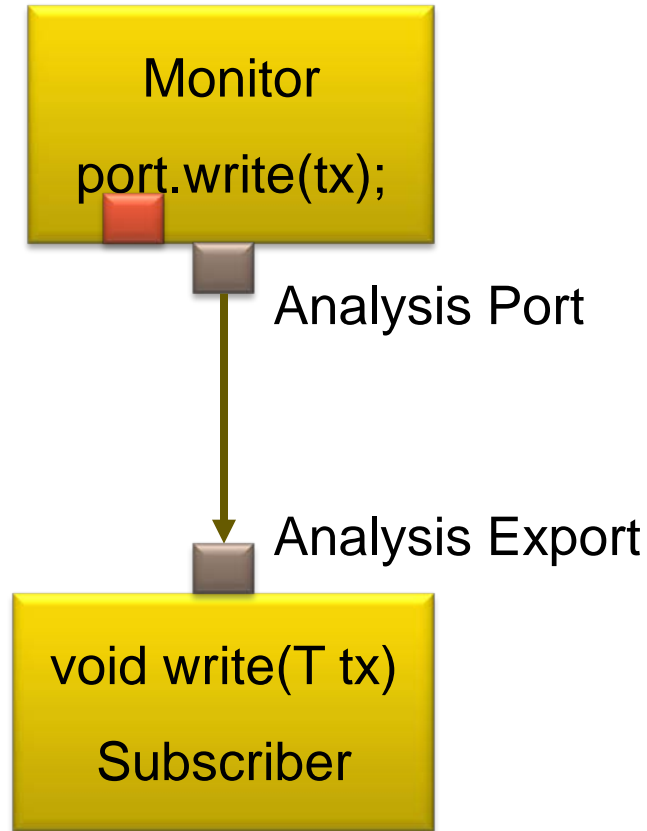
```

```
class my_driver extends ovm_driver #(REQ, RSP);  
...  
virtual task run;  
    REQ req;  
    RSP rsp = new;  
  
    forever  
    begin  
        seq_item_port.get( req ); // from sequencer  
        // Wiggle pins  
        ...  
  
        rsp.set_id_info(req);  
        seq_item_port.put( rsp ); // to sequencer  
  
    end  
endtask: run  
...
```

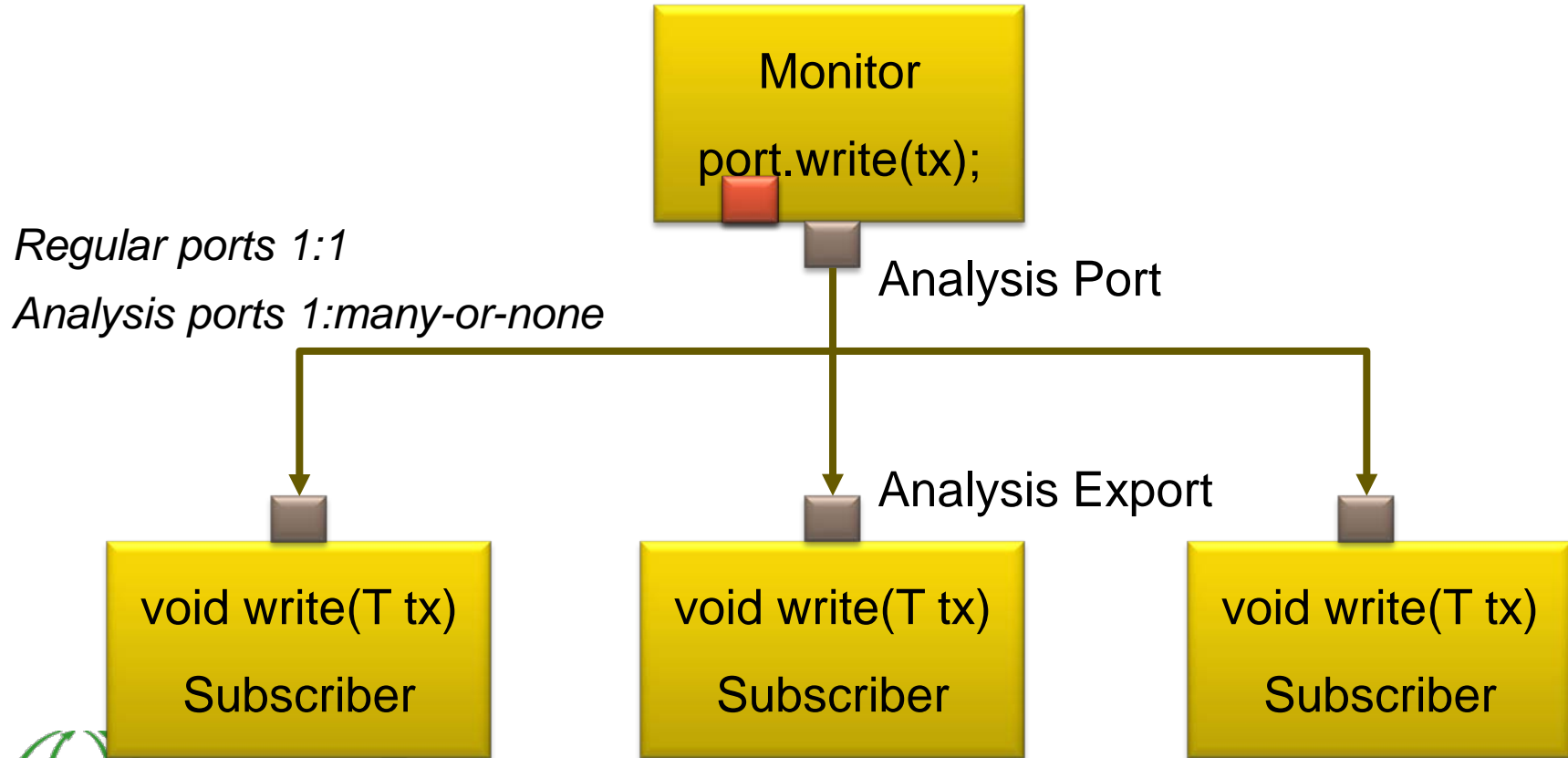
Direct vs Mediated Communication



Regular ports 1:1



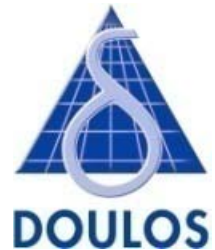
Analysis Ports



The Communication and Customization Mechanisms in OVM and UVM

CONTENTS

- *Introducing Doulos*
- ***Let's get Technical...***
 - *Communication Mechanisms*
 - ***Customization Mechanisms***



```
function void build;
  super.build();
  my_seq_h = my_seq::type_id::create("my_seq_h", this);
  my_driv_h = my_driv::type_id::create("my_driv_h", this);
  my_mon_h = my_mon::type_id::create("my_mon_h", this);
endfunction: build
```

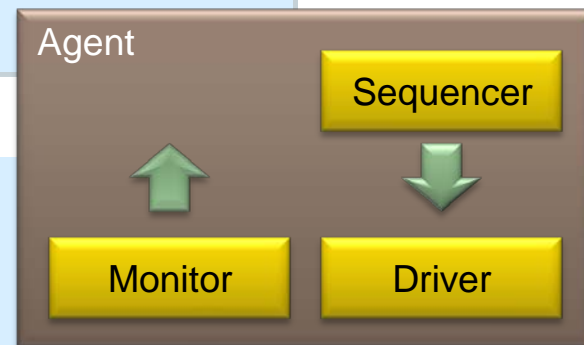


```
class my_test extends uvm_test;
```

...

```
function void build;
  super.build();
  my_driv::type_id::set_type_override( alt_driver::get_type() );
  my_env_h = my_env::type_id::create("my_env_h", this);
endfunction: build
```

...



Replace components, transactions, or sequences

```
class my_driver extends uvm_driver #(my_transaction);
```

```
...
```

```
task run;
```

```
  forever begin
```

```
    my_transaction tx;
```

```
    seq_item_port.get(tx);
```

```
    call_modify(tx);
```

```
    // Wiggle pins of DUT
```

```
    ...
```

```
  end
```

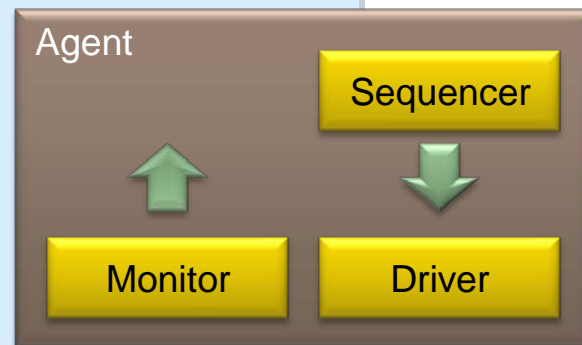
```
endtask: run
```

```
`uvm_register_cb(my_driver, my_cb_interface)
```

```
task call_modify( my_transaction tx );
```

```
  `uvm_do_callbacks( my_driver, my_cb_interface, modify( this, tx ) )
```

```
endtask
```

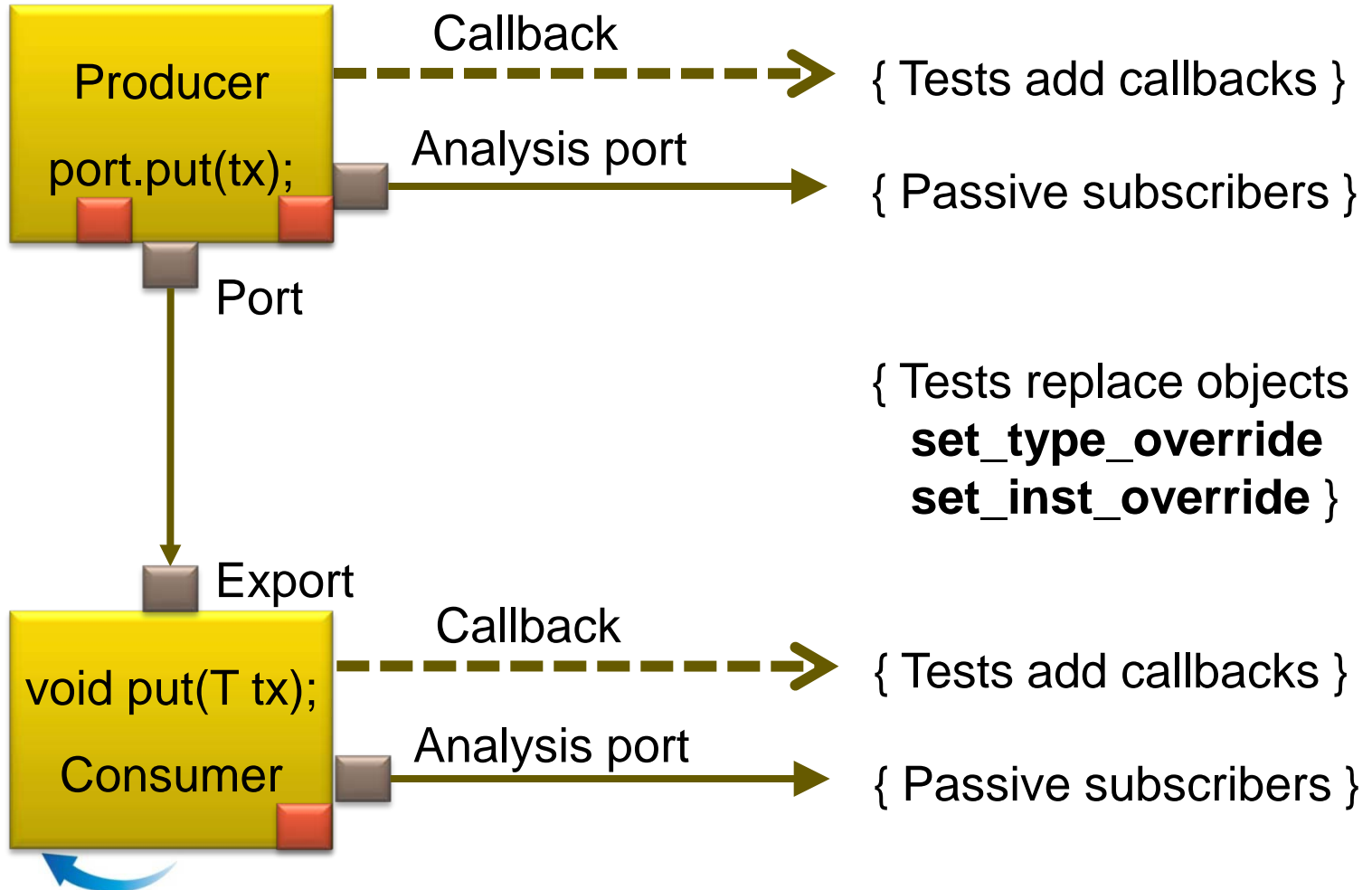


Design callbacks into each component

```
class my_cb extends my_cb_interface;  
...  
virtual task modify( my_driver driver, my_transaction tx );  
    ... modify transaction here  
endtask  
endclass
```

```
class my_test extends uvm_test;  
...  
my_cb my_cb_h;  
...  
function void start_of_simulation;  
    my_driver driver_h;  
    $cast(driver_h, uvm_top.find("*.my_driver_h"));  
    my_cb_h = new("my_cb_h");  
    my_driver_cb_t::add(driver_h, my_cb_h);  
endfunction  
endclass
```

Summary

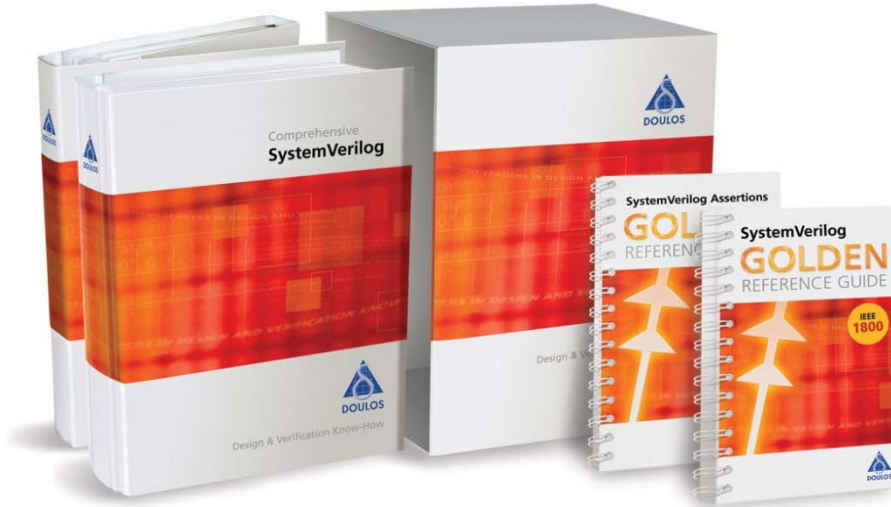


The Communication and Customization Mechanisms in OVM and UVM

CONTENTS

- *Introducing Doulos*
- *Let's get Technical...*
 - *Communication Mechanisms*
 - *Customization Mechanisms*
- ***OVM and UVM Offers from Doulos***





- OVM and UVM Adopter Class
- SystemVerilog Training Customized for OVM / UVM
- Public open-enrolment or private on-site classes



- Comprehensive reference guide to OVM
- Focuses on the most important issues
- Practical hints-and-tips
- 210 pages



Free OVM / UVM Resources

- Visit www.doulos.com

The screenshot shows the Doulos website interface as of Tuesday 8 June 2010. The page features a navigation menu with links for Home, Company, Training, News, KnowHow, and Products. The main header includes the Doulos logo and the tagline "Developing & Delivering KnowHow". A prominent "KnowHow" banner is displayed. On the left, a sidebar menu lists various technologies: VHDL, FPGA, Verilog, SystemC, TLM-2.0, SystemVerilog, OVM, UVM, VMM, PSL, Perl, Tcl/Tk, ARM, and a Video Gallery. The main content area is titled "UVM - The Universal Verification Methodology" and features a large UVM logo with a circular arrow. Below the logo, the text reads "The Universal Verification Methodology" and "At Last, One Functional Verification Methodology for Everyone!". A paragraph states: "The UVM 1.0 Early Adopter release was made available by Accellera for download on 17 May 2010 with the explicit endorsement of all the major simulator vendors. Now, at last, we have one single SystemVerilog verification methodology that really is supported by all EDA vendors." On the right side, there are two sections: "Training" which includes the UVM logo and the text "Training from Doulos for UVM.", and "Related Courses" which lists: UVM Adopter Class, OVM Adopter Class, Comprehensive SystemVerilog, and Modular SystemVerilog. A "Doulos SystemVerilog Training Programs" section also mentions "Now incorporating OVM".





Delivering Know-How

www.doulos.com

Hardware Design

- » VHDL » Verilog » SystemVerilog
- » Actel » Altera » Xilinx

Embedded Systems and ARM

- » C » C++ » UML » RTOS » Linux
- » ARM Cortex A/R/M series

ESL & Verification

- » SystemC » TLM-2.0 » SystemVerilog
- » OVM/VMM/UVM » Perl » Tcl/Tk