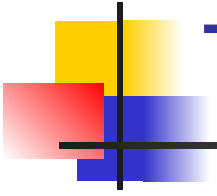
UVM : TLM

new
pre_randomize
post_randomize

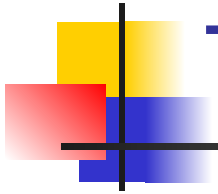
sub_class_valid.constraint_mode(0);
In environment test, set drain time of 1us



TLM Interfaces

UVM provides a collection of classes and interfaces for transaction-level modeling (TLM).

- enable transaction-level communication between entities
- requests are sent and responses received by transmitting transaction objects through various interfaces.
- UVM TLM 1 is concerned with passing messages of arbitrary types through ports and exports.
- UVM TLM 2 is concerned with modeling protocols and is based on sockets and a standardized transaction object called a generic payload.
- Sockets provide both blocking and non-blocking style of communication as well as forward and backward paths.



TLM 1 – 3 types

Blocking

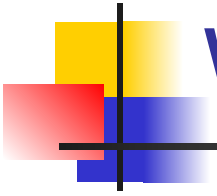
A blocking interface conveys transactions in blocking fashion ; its methods do not return until the transaction has been successfully sent or retrieved. Its methods are defined as tasks.

Non-blocking

A non-blocking interface attempts to convey a transaction without consuming simulation time. Its methods are declared as functions. Because delivery may fail (e.g. the target component is busy and can not accept the request), the methods may return with failed status.

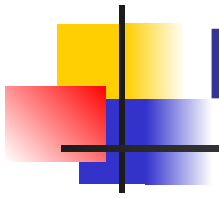
Combined

A combination interface contains both the blocking and non-blocking variants.



Why TLM?

Higher level of abstraction
Re-usable; plug and play
Easier to maintain
Less Code
Easier to implement
Faster Simulation
Standard connection to SystemC
Able to be used for reference model development.



Methods

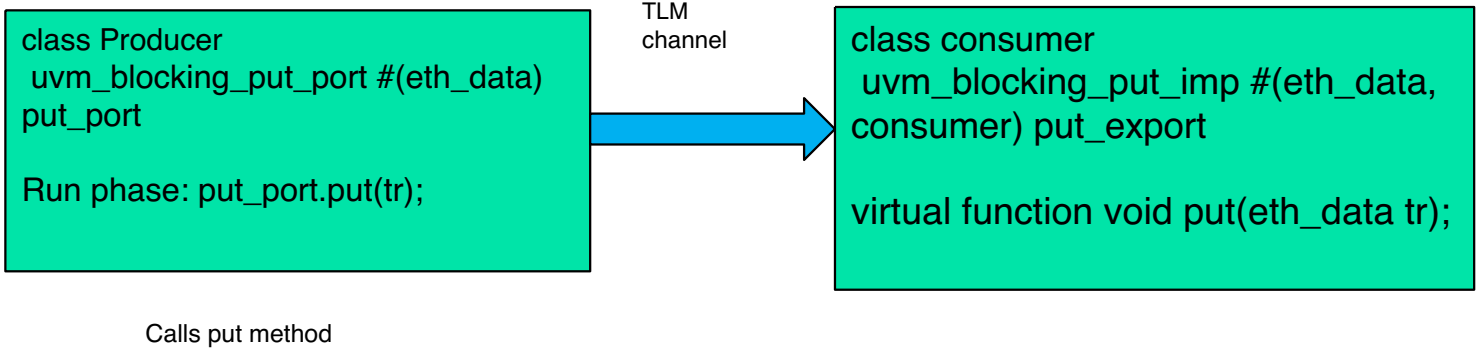
blocking

- virtual task put(input T1 t)
- virtual task get(output T2 t)
- virtual task peek(output T2 t)

Non-blocking

- virtual function bit try_put(input T1 t)
- virtual function bit can_put()
- virtual function bit try_get(output T2 t)
- virtual function bit can_get()
- virtual function bit try_peek(output T2 t)
- virtual function bit can_peek()

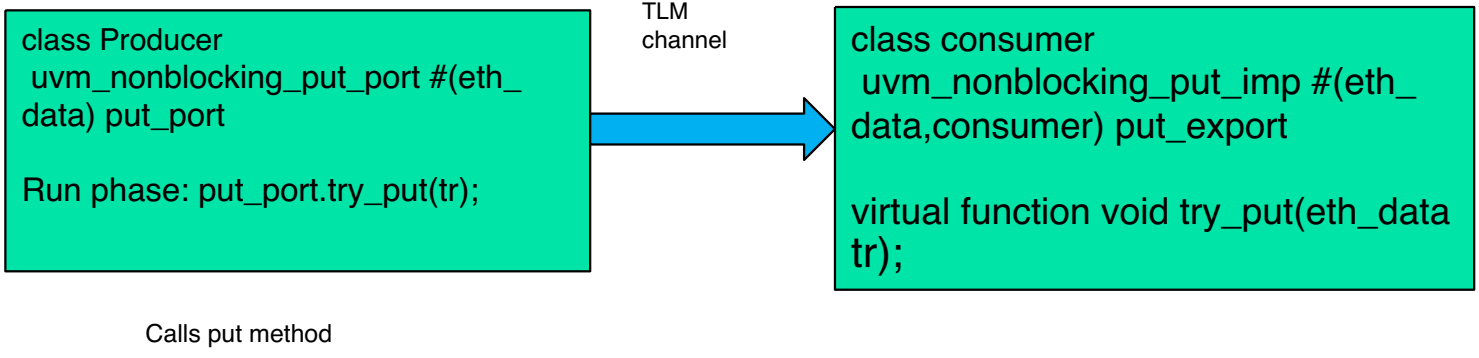
PUSH method (blocking)




```
class environment
Producer p1;
Consumer c1;

P1.put_port.connect(c1.put_export);
```

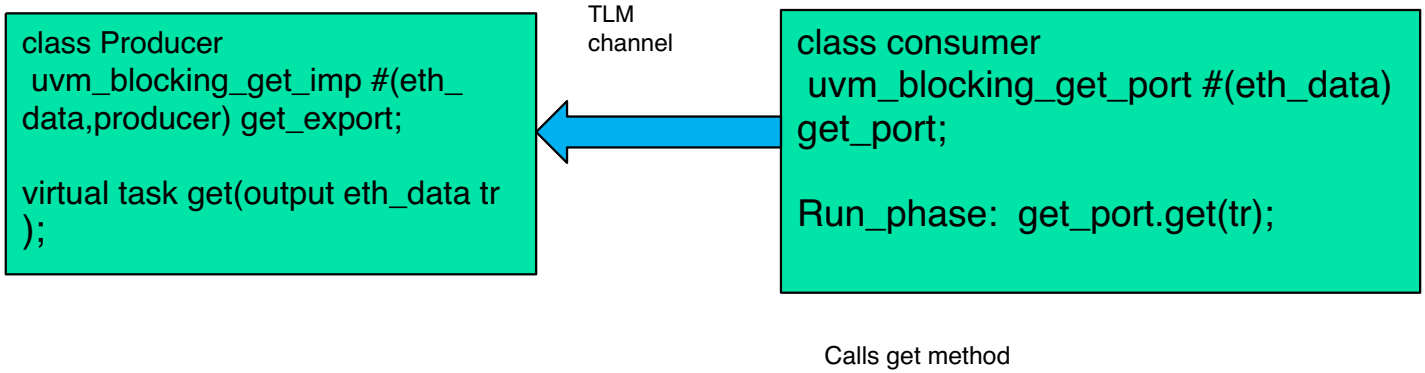
PUSH method (non-blocking)




```
class environment
Producer p1;
Consumer c1;

P1.put_port.connect(c1.put_export);
```

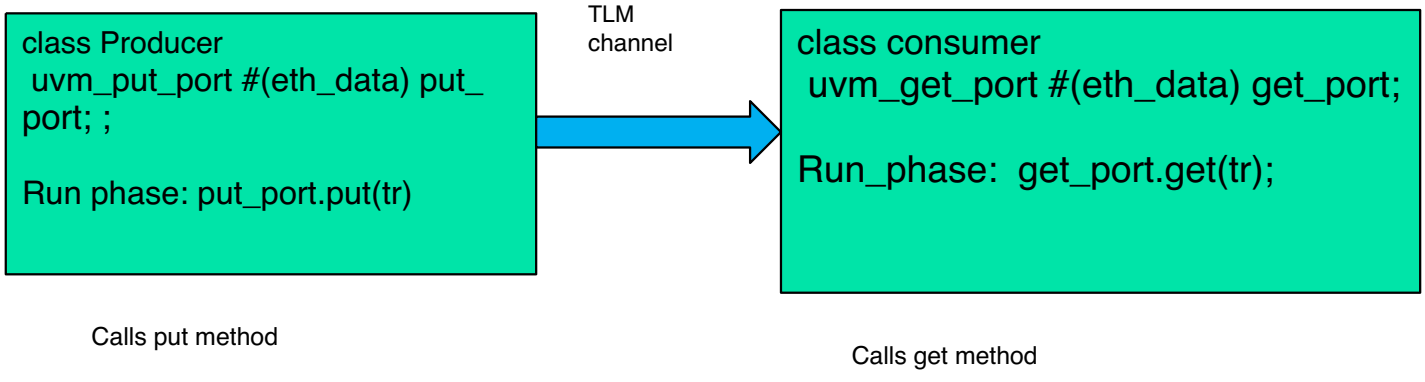
PULL method




```
class environment
Producer p1;
Consumer c1;

 c1.get_port.connect(p1.get_export);
```

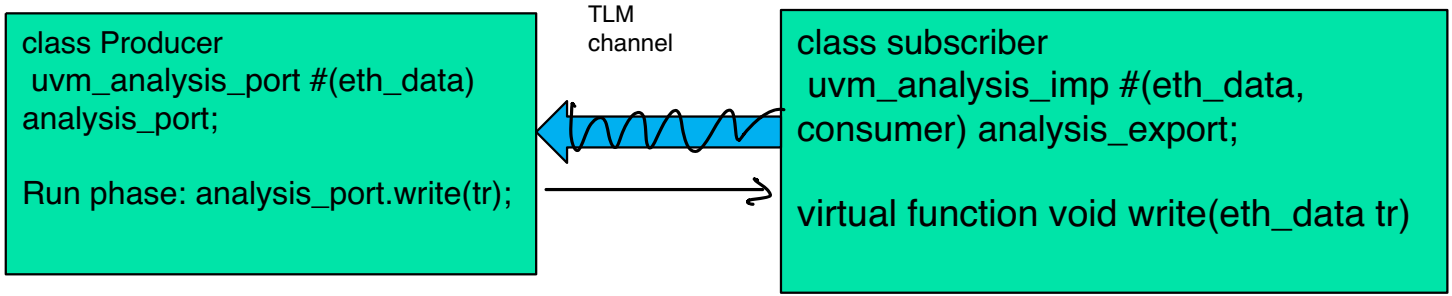
FIFO method




```
class environment
Producer p1;
Consumer c1;

  p1.put_port.connect(tr_fifo.put_export);
  c1.get_port.connect(tr_fifo.get_export);
```

Analysis method



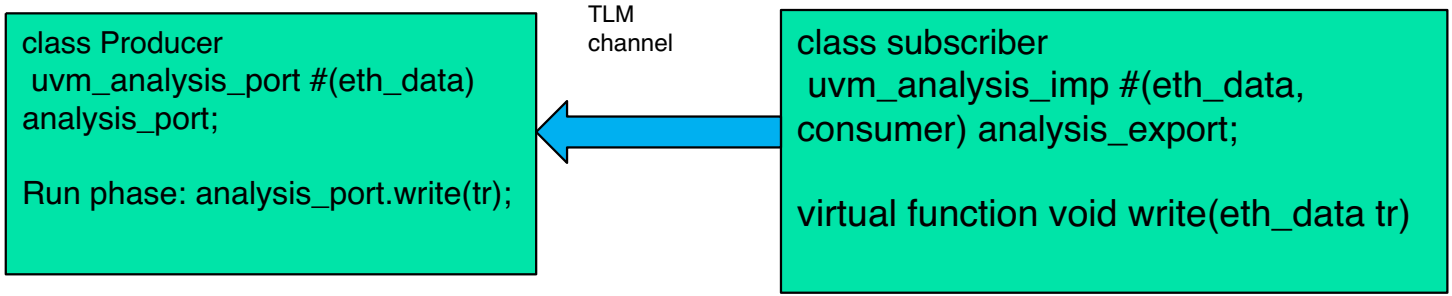
Calls write method

```
class environment
Producer p1;
Subscriber c1;

 p1.analysis_port.connect(c1.analysis_export);
```

Note: Can be left unconnect, can have multiple subscribers, also multiple variants

Analysis method



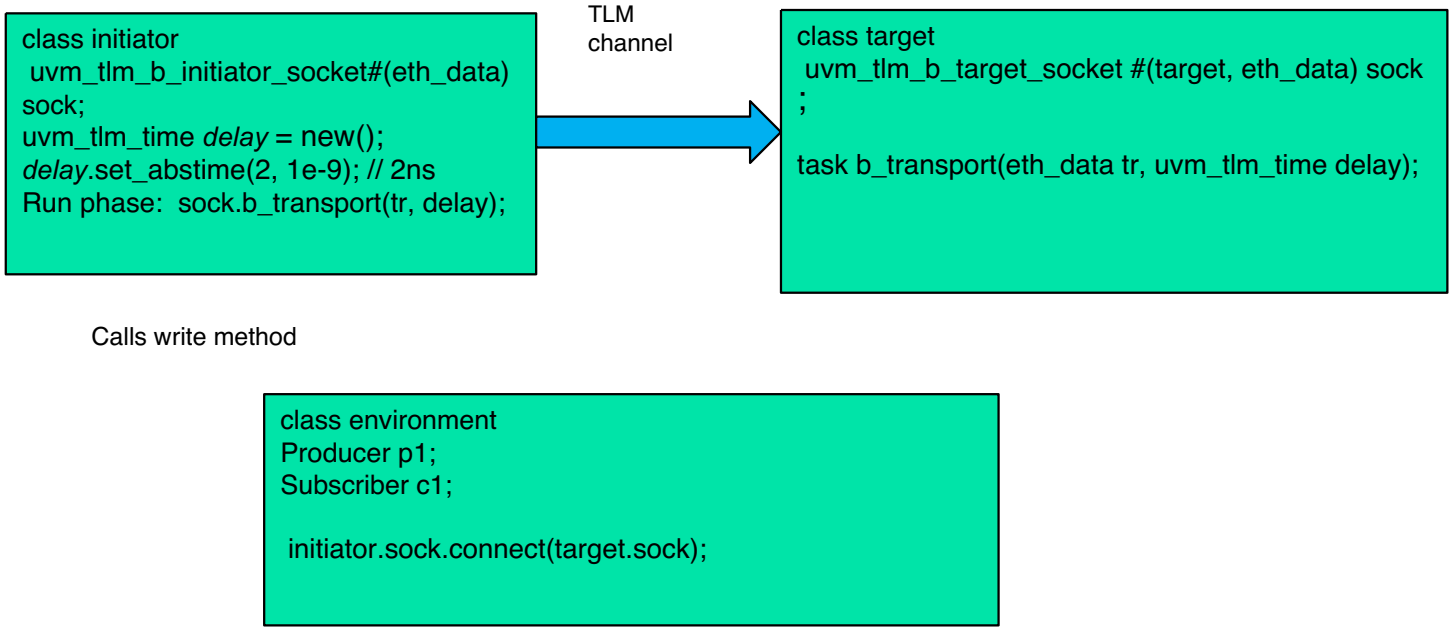
Calls write method

```
class environment
Producer p1;
Subscriber c1;

 p1.analysis_port.connect(c1.analysis_export);
```

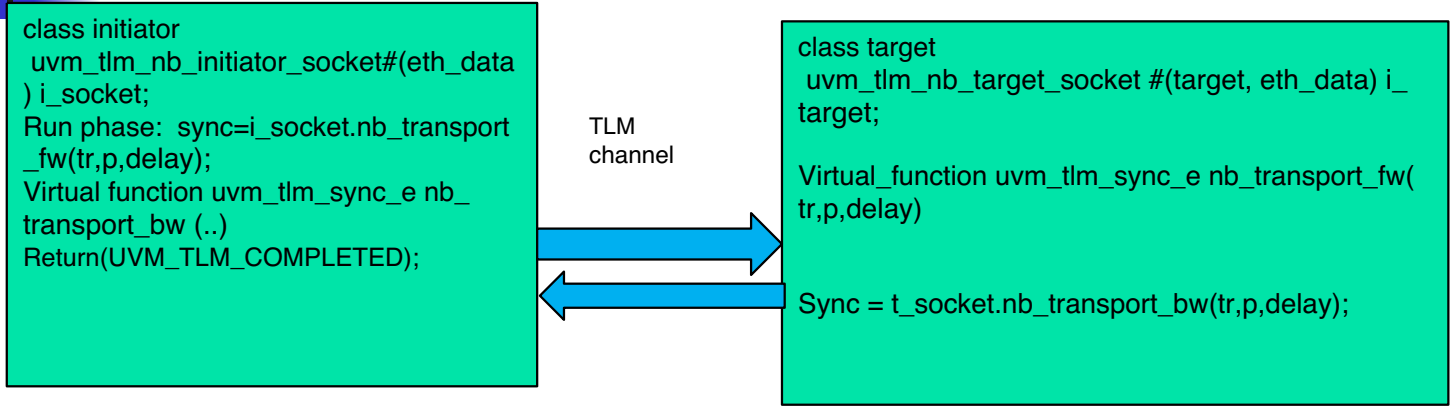
Note: Can be left unconnect, can have multiple subscribers, also multiple variants

TLM2 : blocking transport



Note: Can be left unconnect, can have multiple subscribers, also multiple variants

TLM2 : non-blocking transport



Calls write method

```
class environment
Producer p1;
Subscriber c1;

 initiator.sock.connect(target.sock);
```

Note: Can be left unconnected, can have multiple subscribers, also multiple variants

UVM : hints

How to generate bad payloads ?

Override eth_data class

```
class eth_data_bad extends eth_data;

function void pre_randomize();
     eth_data_valid.constraint_mode(0);
endfunction
 `uvm_object_utils_begin(eth_data_bad)
 `uvm_object_utils_end
function eth_data_bad::new(string name = "Trans");
   super.new(name);
endfunction: new

endclass: eth_data_bad


 set_type_override_by_type (eth_data::get_type(), eth_data
_bad::get_type());
```

Test end too early?

Test ends before all the TX data is received by the monitor

```
Class environment_test …

virtual task main_phase(uvm_phase phase);
    uvm_objection objection;
    super.main_phase(phase);
    objection = phase.get_objection();
    objection.set_drain_time(this,1us);
  endtask
```