Exploiting the TLM-2 Features of VMM 1.2

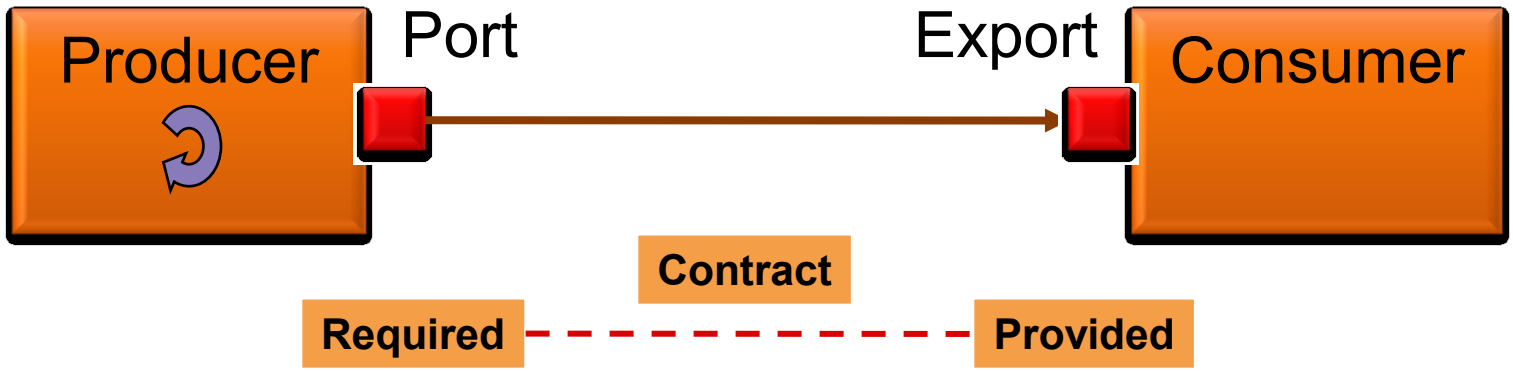
John Aynsley and Doug Smith

Doulos

Exploiting the TLM-2 Features of VMM 1.2

- TLM principles and benefits
- Ports, exports and binding
- Transport and analysis interfaces
- Generic payload
- Guidelines
- The TLI

Required / Provided Interfaces



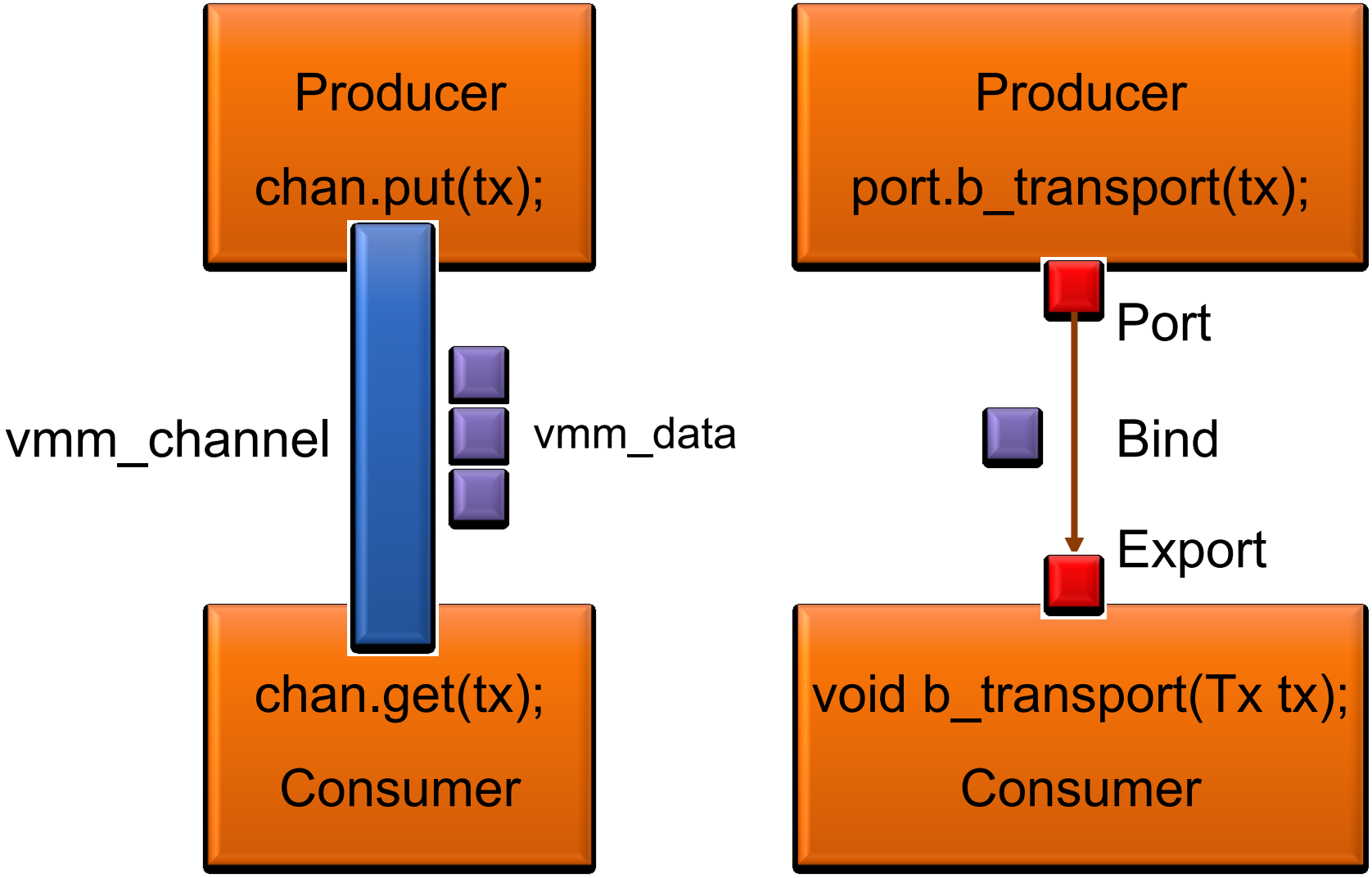

```
class producer extends vmm_xactor;
  vmm_tlm_b_transport_port #(producer, transaction) port;
  // Calls b_transport
  ...
```

```
class consumer extends vmm_xactor;
  vmm_tlm_b_transport_export #(consumer, transaction) export;
  // Implements b_transport
  ...
```

```
prod.port.tlm_bind( cons.export );
```

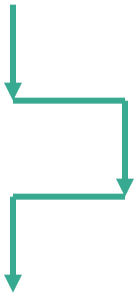
Connection is type-safe

Mediated vs Direct Communication

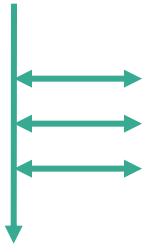


Blocking and Non-blocking Calls

- Terminology borrowed from SystemC TLM:



  - A blocking method may suspend (execute a procedural timing control)
  - A blocking method must be a SystemVerilog task
  - A blocking method always succeeds



  - A non-blocking method must not suspend
  - A non-blocking method must be a SystemVerilog function
  - A non-blocking method returns a status flag (success/failure)

  - VMM 1.2 adds b_transport and nb_transport_*

Benefits of using TLM within VMM

- Direct communication is fast – no intermediate channel
- Simple completion model – independent of consumer
- Simple type-safe connections
- SystemC TLM-2.0 is an industry standard –

  better interoperability

Blocking Transport Port

```
class my_tx extends vmm_data;
  ...
```

```
class producer extends vmm_xactor;
  vmm_tlm_b_transport_port #(producer, my_tx) port;
  ...
  begin: loop
    my_tx  tx;
    int        delay;
    assert( randomized_tx.randomize() )
        else `vmm_error(log, "tx.randomize() failed");
    $cast(tx, randomized_tx.copy());
    port.b_transport(tx, delay);
    ...
```

Transaction completed in a single method call

Blocking Transport Export

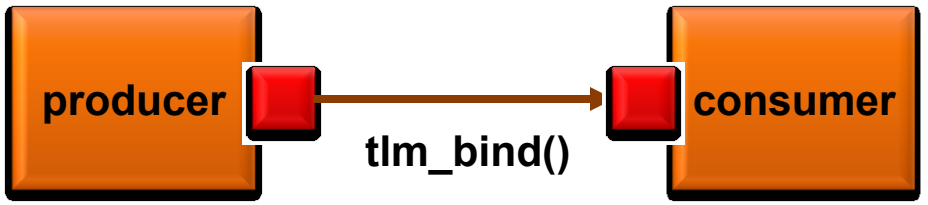
```
class consumer extends vmm_xactor;
  vmm_tlm_b_transport_export #(consumer, my_tx) export;
  ...
  virtual function void build_ph;
    export = new(this, "export");
  endtask

  task b_transport(int id = -1, my_tx trans, ref int delay);
    int addr = trans.addr;
    int data = trans.data;
    ...
    #(delay);
  endtask
  ...
```

Method may block

Return only when transaction is complete

Port-to-Export Binding

```
class tb_env extends vmm_group;
  producer   m_producer;
  consumer  m_consumer;

  virtual function void build_ph;
    m_producer  = new( " m_producer ", this );
    m_consumer = new( " m_consumer ", this );
  endfunction

  virtual function void connect_ph;
    m_producer.port.tlm_bind( m_consumer.export );
  endfunction
  ...
```



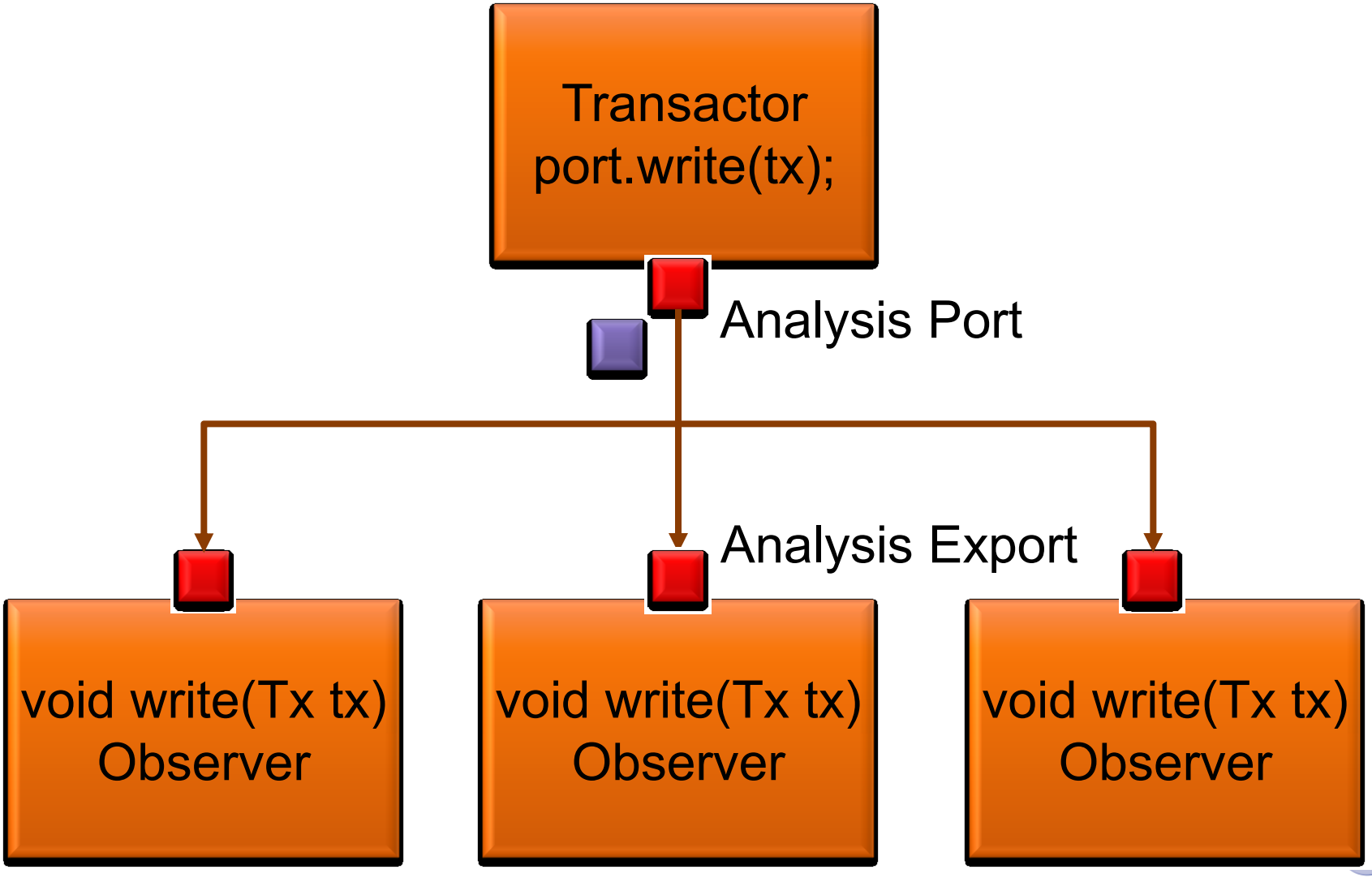
The Generic Payload

Attribute	Type	Modifiable?
Command	tlm_command	No
Address	longint	Interconnect only
Data	byte[]	Yes (read command)
Data length	int unsigned	No
Byte enable pointer	byte[]	No
Byte enable length	int unsigned	No
Streaming width	int unsigned	No
DMI hint	bit	Yes
Response status	tlm_response_status	Target only
Extensions	vmm_tlm_extension_base[ ]	Yes

Use off-the-shelf for abstract memory-mapped bus modeling

Extend to model specific protocols

Analysis Ports



Binding an Analysis Port

```
class  transactor  extends  vmm_xactor;
  vmm_tlm_analysis_port #(transactor, my_tx)  ap;
  ...
  virtual task main;
    my_tx  tx;
    ...
    ap.write(tx);
```

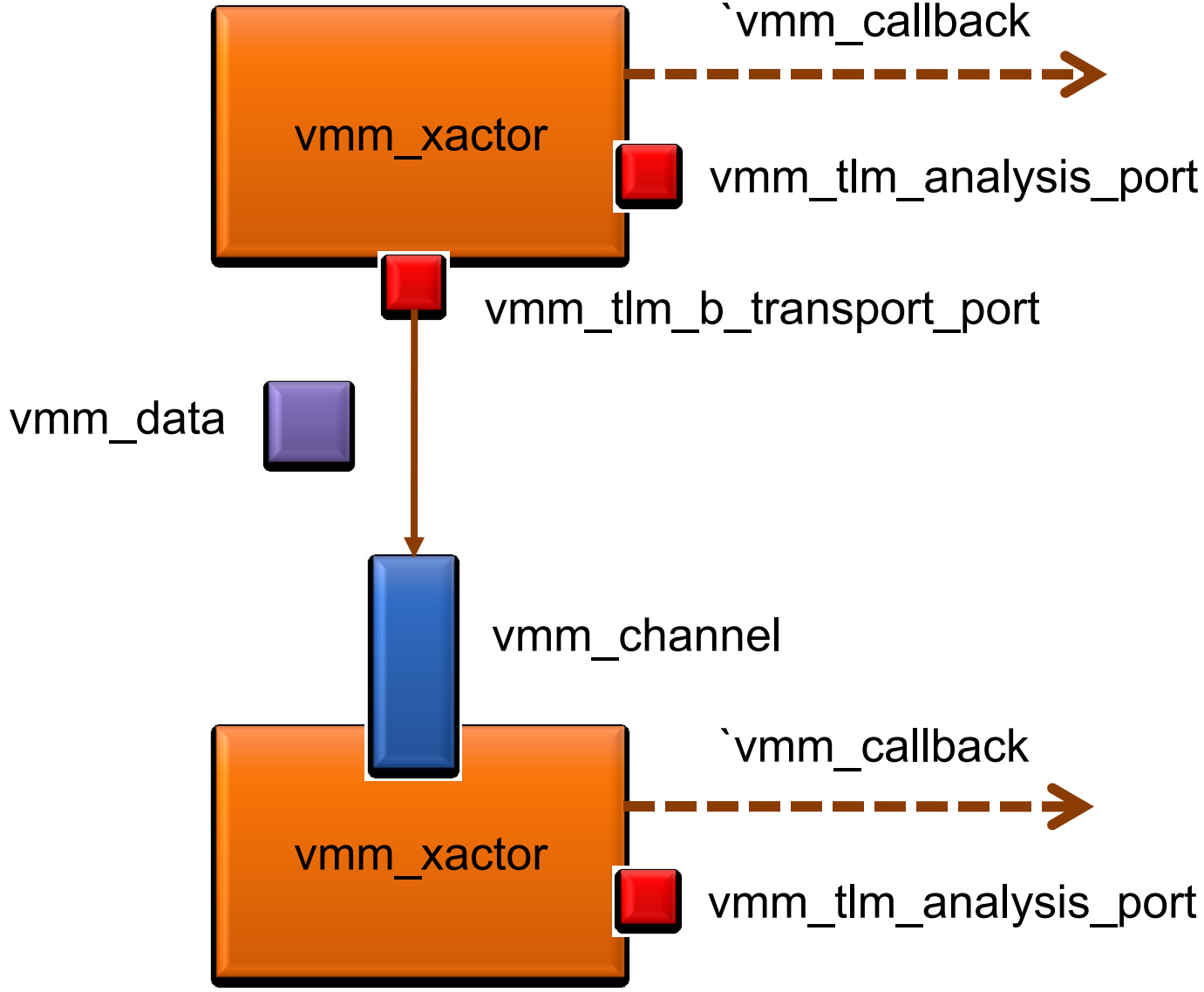
```
class observer extends vmm_object;
  vmm_tlm_analysis_export  #(observer, my_tx) export;
  ...
  function void write(int id, my_tx tx);
    ...
```

Cannot modify transaction

```
class tb_env extends vmm_group;
  ...
  virtual function void connect_ph;
    m_transactor.ap.tlm_bind( m_observer_1.export );
    m_transactor.ap.tlm_bind( m_observer_2.export );
    ...
```

Cannot control call order

Communication Options



VMM 1.2 Guidelines

- **Blocking transport port** - preferred for master-like transactors
- **Channel** – preferred for slave-like transactors
- **Analysis port** – preferred for sending transactions to passive objects
- **Callbacks** – preferred when modifying transactions for specific tests
- **Notifications** – preferred for data-less synchronization between transactors
- **Non-blocking transport** – only for communication with SystemC

Example: Transaction

```
class  my_tx  extends  vmm_data;
  rand int addr;
  rand int data;
  constraint  c_addr  { addr >= 0; addr < 256; }
  constraint  c_data  { data >= 0; data < 256; }
  `vmm_typename(my_tx)
  `vmm_class_factory(my_tx)
  `vmm_data_member_begin(my_tx)
     `vmm_data_member_scalar(addr, DO_ALL)
     `vmm_data_member_scalar(data, DO_ALL)
  `vmm_data_member_end(my_tx)
endclass: my_tx
```

```
typedef  vmm_channel_typed #(my_tx)  my_channel;
```

Example: Producer

```
class  my_gen  extends  vmm_xactor;
  vmm_tlm_b_transport_port  #(my_gen,  my_tx)  m_port;
  vmm_tlm_analysis_port       #(my_gen,  my_tx)  m_ap;
  ...
  begin: loop
    assert( randomized_tx.randomize() ) ...
    $cast(tx, randomized_tx.copy());

    `vmm_callback(my_gen_callbacks,  pre_trans(this, tx));
    m_port.b_transport(tx, delay);
    `vmm_callback(my_gen_callbacks,  post_trans(this, tx));
    m_ap.write(tx);
  end
  ...
```

Callbacks at significant points

Analysis port after callback for monitoring

Example: Consumer

```
class  my_bfm  extends  vmm_xactor;
  my_channel  m_chan;
  vmm_tlm_analysis_port #(my_bfm, my_tx)  m_ap;
  ...
  begin: loop
    `vmm_callback(my_bfm_callbacks, pre_trans(this, tx));
    m_chan.activate(tx);
    m_chan.start();
    @(i_f.bus_cb); ...
    m_chan.complete();
    m_chan.remove();
    `vmm_callback(my_bfm_callbacks, post_trans(this, tx));
    m_ap.write(tx);
  end
```

Blocking

Causes notification vmm_data::STARTED

Causes notification vmm_data::ENDED

Removes tx from channel

Example: Env

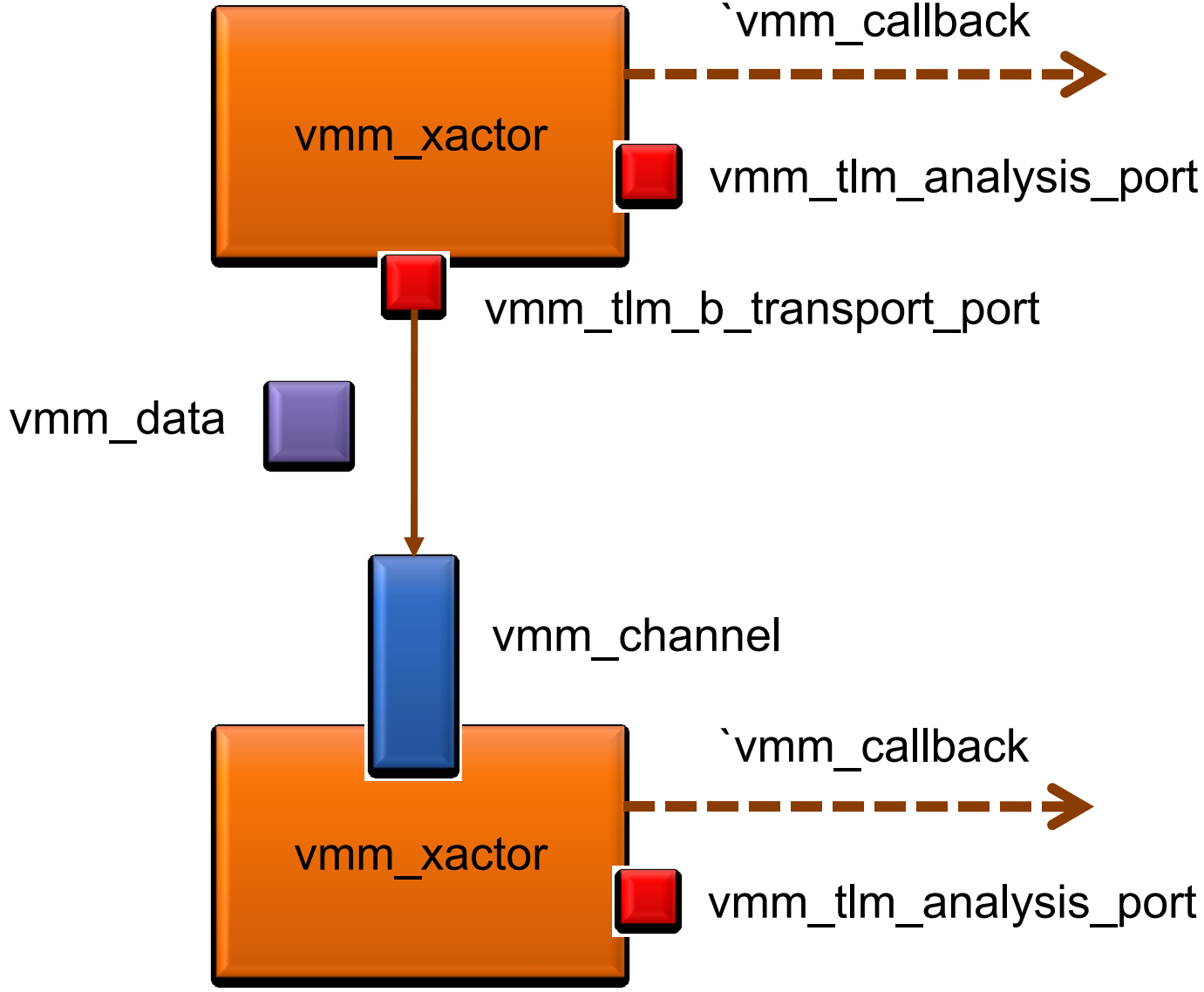
```
class tb_env extends vmm_group;
  virtual function void build_ph;
    m_tx_chan  = new( "my_channel", "m_tx_chan" );
    m_tx_chan.reconfigure(1);
    m_gen      = new( "m_gen", this );
    m_bfm      = new( "m_bfm", this );
  endfunction

  function void connect_ph();
    vmm_connect #(.D(my_tx))::tlm_bind(m_tx_chan, m_gen.m_port,
                                       vmm_tlm::TLM_BLOCKING_EXPORT );
    m_bfm.m_chan = m_tx_chan;
  endfunction
  ...
```

Input channel full level = 1

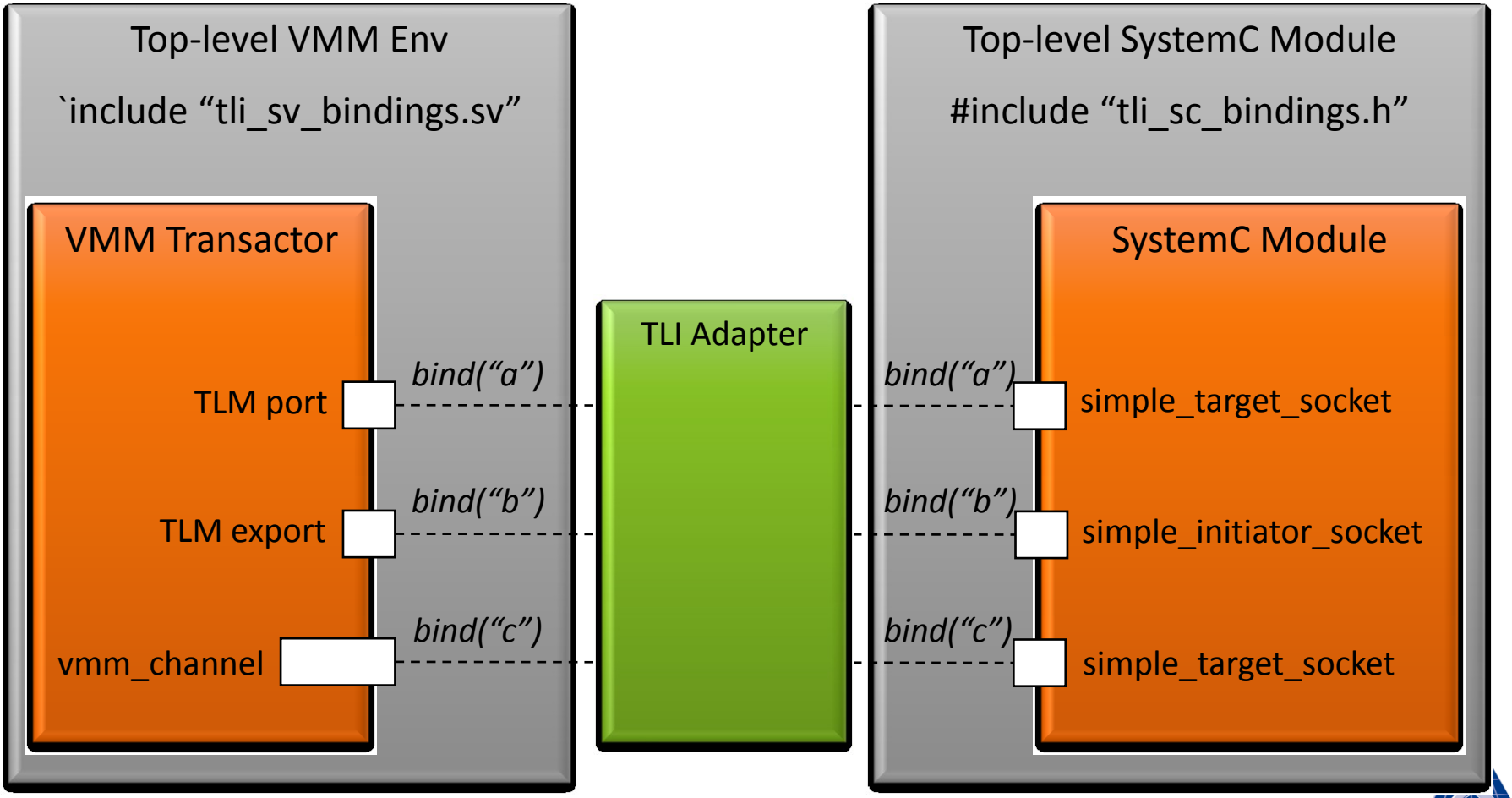
b_transport_port-to-channel

Consumer input channel

Communication Options - Summary



Transaction-Level Interface (TLI)



TLI - Kinds of Communication

- SystemVerilog blocking port, SystemC LT target
- SystemVerilog non-blocking port, SystemC AT target
- SystemVerilog vmm_channel, SystemC LT target
- SystemVerilog vmm_channel, SystemC AT target
- SystemVerilog analysis port, SystemC analysis subscriber

SV-to-SC

VMM ports or channels

Generic or user-defined payload

- SystemC LT initiator, SystemVerilog blocking export
- SystemC AT initiator, SystemVerilog non-blocking export
- SystemC LT initiator, SystemVerilog vmm_channel
- SystemC AT initiator, SystemVerilog vmm_channel
- SystemC analysis port, SystemVerilog analysis subscriber

SC-to-SV

LT or AT

TLI Example – VMM Side

```
class my_xactor extends vmm_xactor;
  vmm_tlm_b_transport_port  #(my_xactor, vmm_tlm_generic_payload)  m_b_port;
  ...
  m_b_port.b_transport(tx, delay);
```

```
`include "tli_sv_bindings.sv"
import vmm_tlm_binds::*;

class  my_env  extends  vmm_group;
  my_xactor  m_xactor;
  function  void  connect_ph();
    tli_tlm_bind( m_xactor.m_b_port, vmm_tlm::TLM_BLOCKING_EXPORT, "sv_tlm_lt");
    ...
```

TLI Example – SystemC Side

```
struct  scmod:  sc_module
{
  tlm_utils::simple_target_socket<scmod>   targ_socket_lt;
  ...
  targ_socket_lt.register_b_transport (this, &scmod::b_transport);
```

```
#include "tli_sc_bindings.h"
struct sctop: sc_module
{
  scmod *m_scmod;
  SC_CTOR(sctop)  {
    m_scmod = new scmod("m_scmod");
    tli_tlm_bind_target (m_scmod->targ_socket_lt, LT, "sv_tlm_lt");
    ...
```

Exploiting the TLM-2 Features of VMM 1.2 – Summary

- TLM principles and benefits
- Ports, exports and binding
- Transport and analysis interfaces
- Generic payload
- Guidelines
- The TLI

System Design

SystemC
ARM • C++

Verification Methodology

e **• PSL • SCV**
SystemVerilog

Hardware Design

VHDL • Verilog
Altera • Xilinx
Perl • Tcl/Tk