2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

UVM and SystemC Transactions – An Update

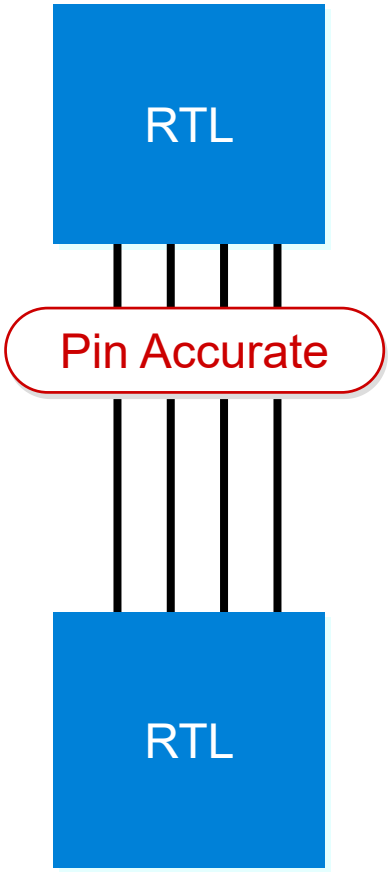
Dr David Long, Doulos

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

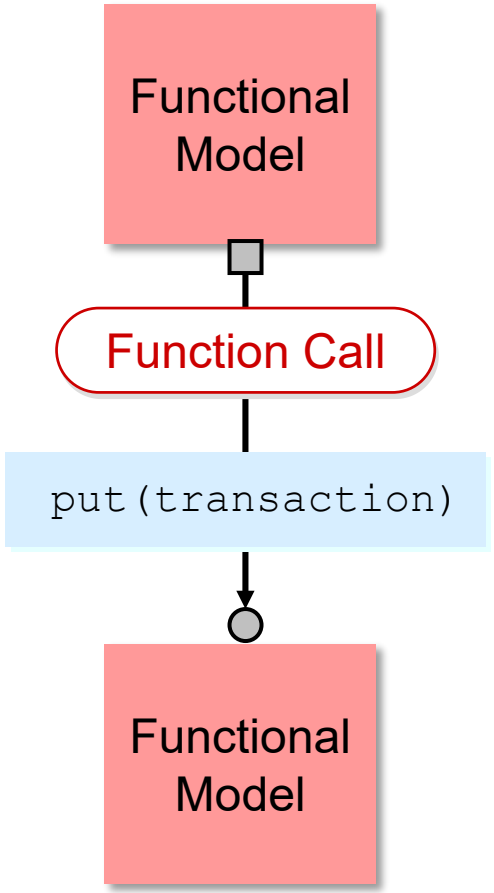
Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions

Transaction-Level Communication

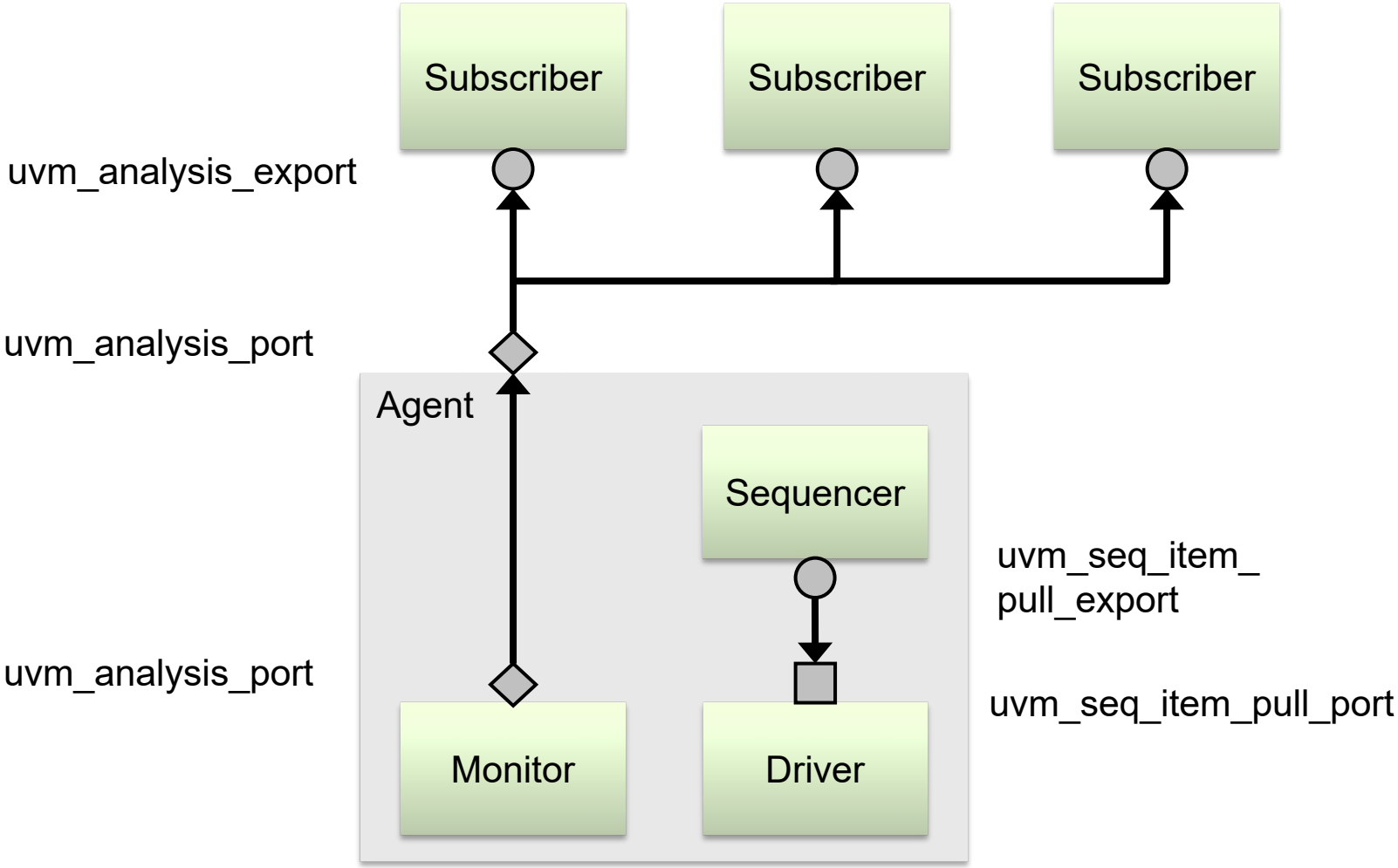


Simulate every event!



100-10,000 X faster simulation!

Transactions within UVM



UVM Execution Phases

- build
- connect
- end_of_elaboration
- start_of_simulation
- run
- extract/check/report
- final

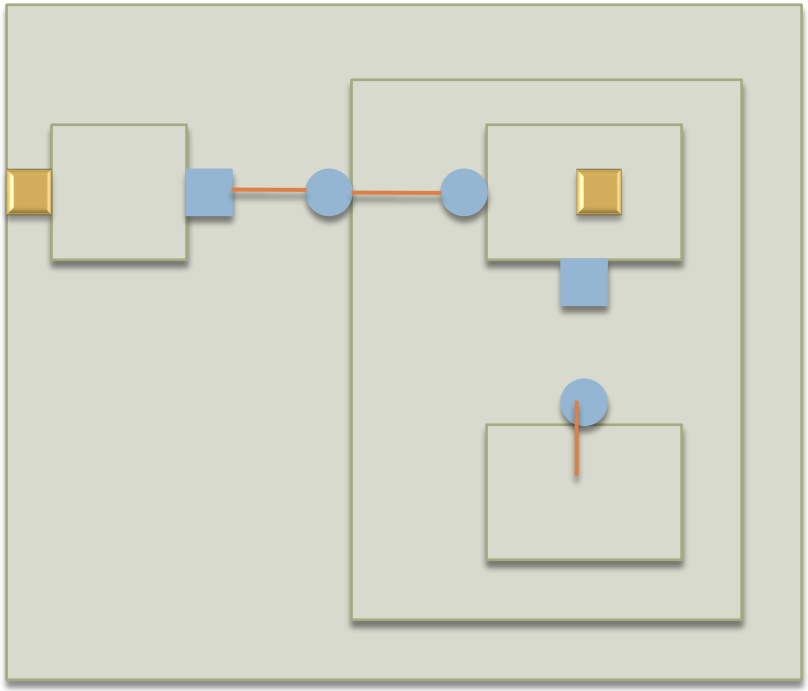
pre_reset
reset
post_reset

pre_configure
configure
post_configure

pre_main
main
post_main

pre_shutdown
shutdown
post_shutdown

A consistent temporal structure

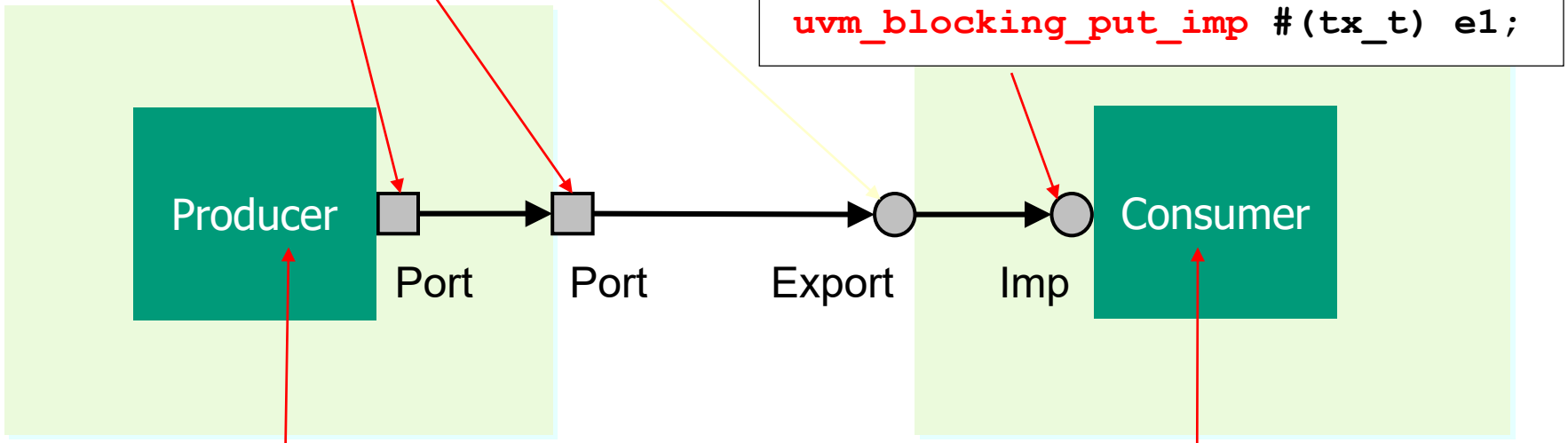


TLM1 Ports, Exports and Imps

```
uvm_blocking_put_port #(tx_t) p1;
```

```
uvm_blocking_put_export #(tx_t) e1;
```

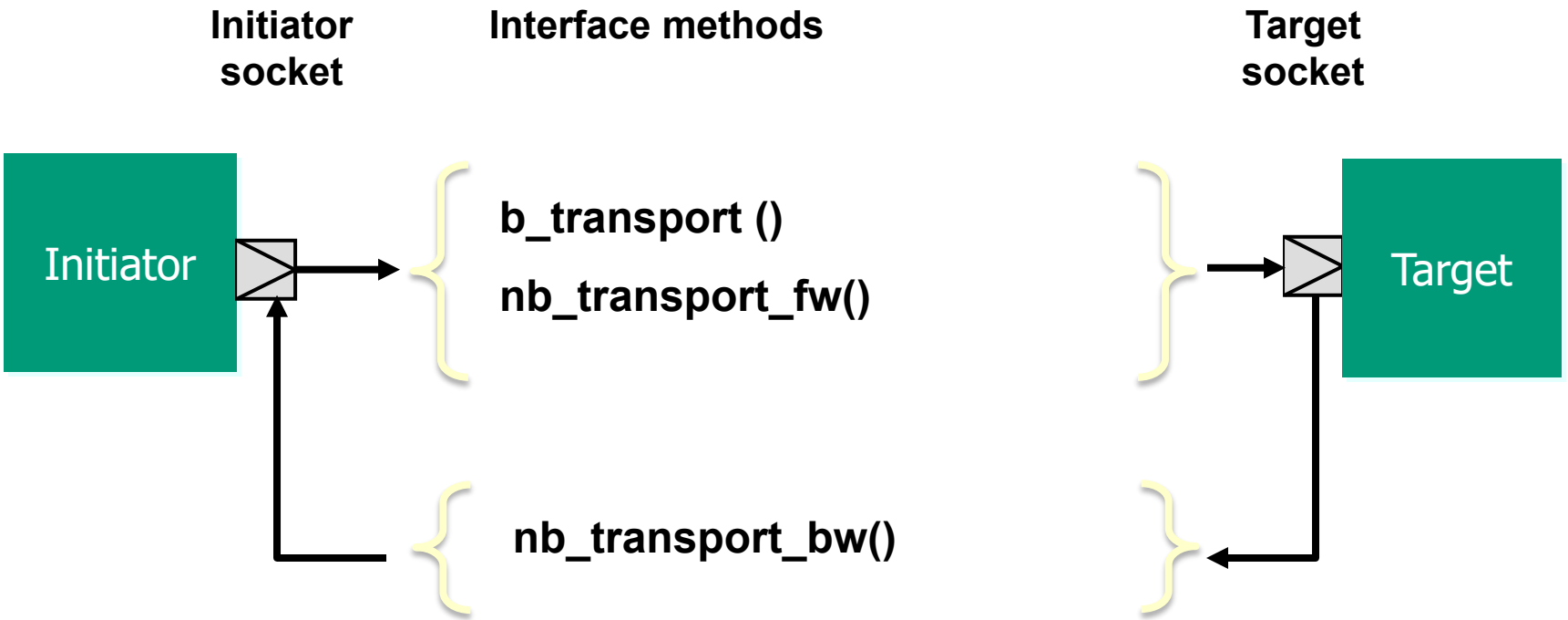
```
uvm_blocking_put_imp #(tx_t) e1;
```




```
p1.put(tx);
```

```
task put(input my_tx tx);
  ...
endtask
```

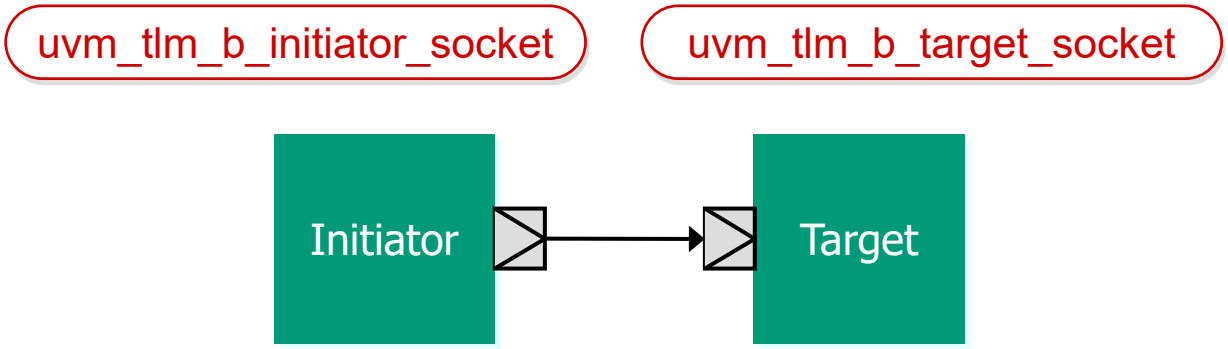
TLM-2.0 Sockets



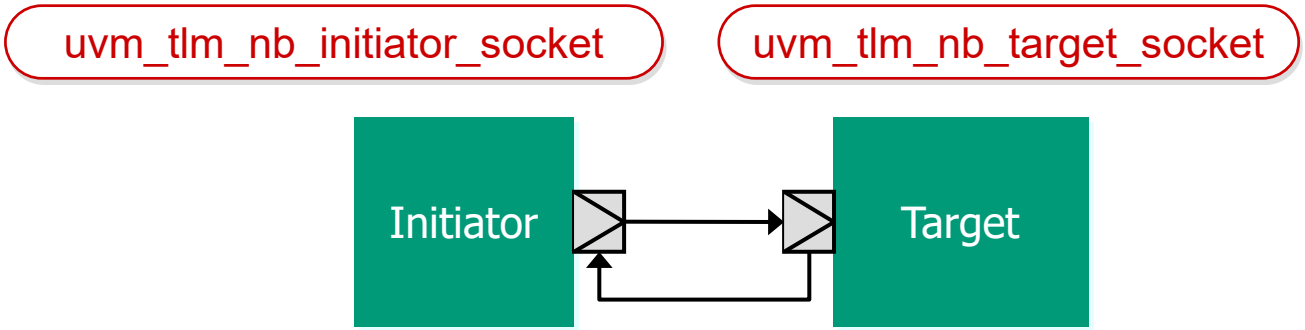
- transactions use generic_payload
- nb_transport_bw only called when nb_transport_fw used
- SystemC sockets support both blocking and non-blocking

UVM Initiator and Target Sockets

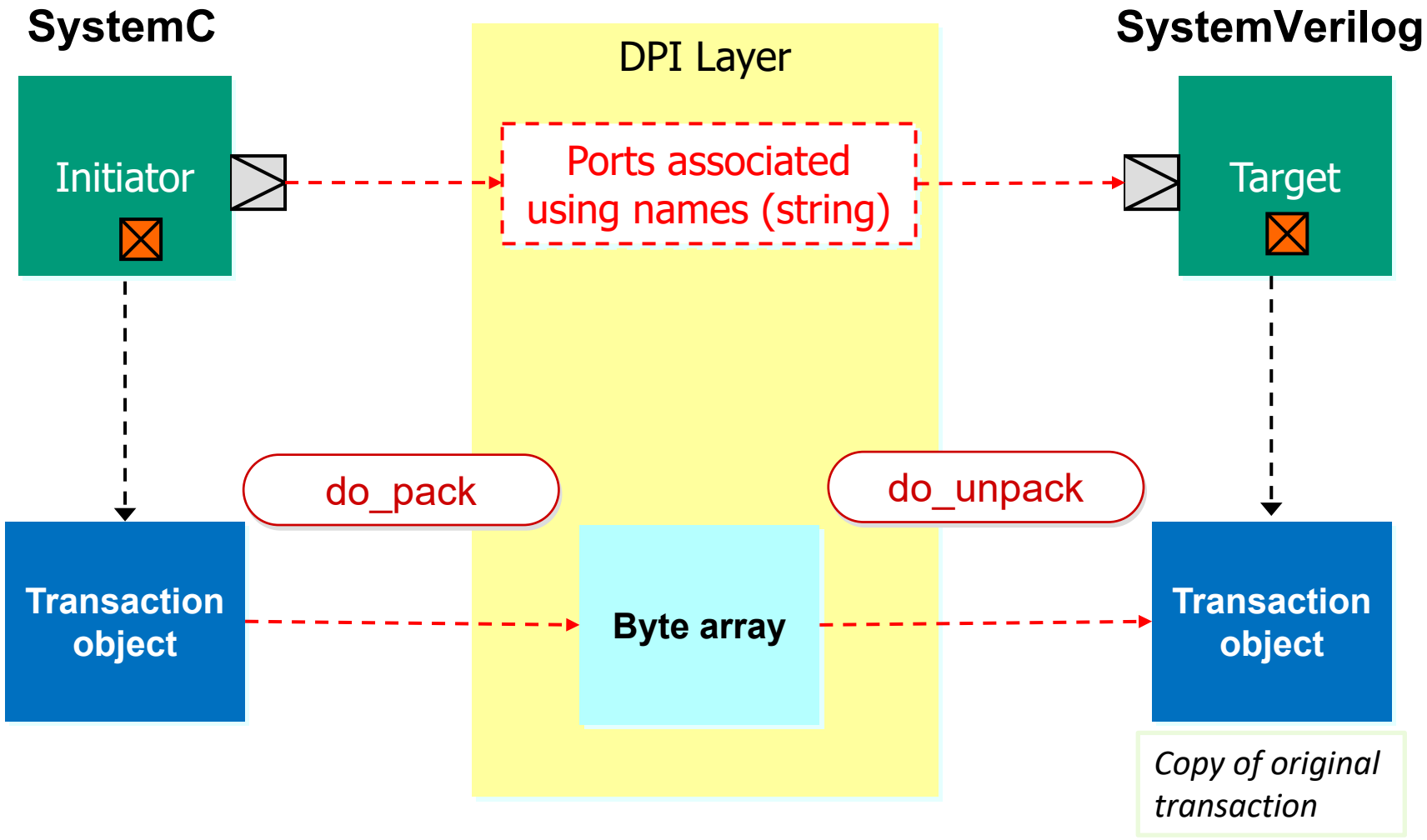
- Blocking



- Non-blocking



Cross-Language Transactions



2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- TLM in UVM and SystemC
- **UVM Connect**
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions

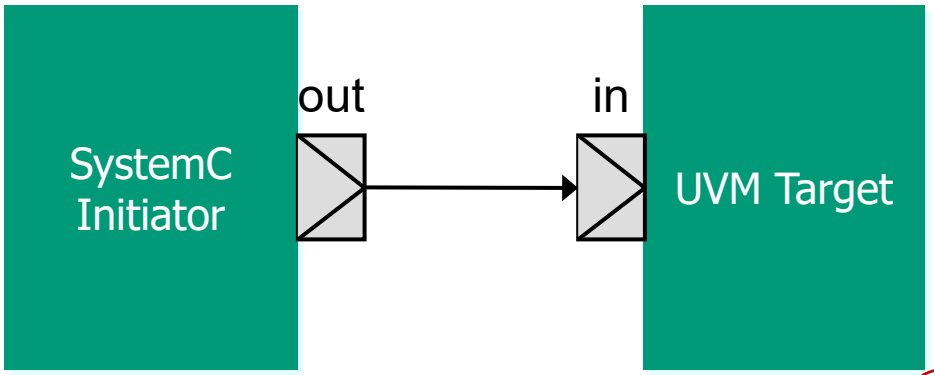
UVM Connect

- Library of classes and functions that enable **connection** and **synchronization** of SystemVerilog and SystemC components via
  - TLM1 ports and exports
  - TLM-2.0 sockets using the generic payload
  - Analysis ports and exports
- Automatic and user-defined converters for transaction classes
- Based on standard SystemVerilog DPI-C so works with simulators from multiple vendors
- Developed by Mentor Graphics but released under Apache 2.0 License with all source code

TLM Connections with UVM Connect

SystemC

```
uvmc_connect(initiator.out,"socket");
```



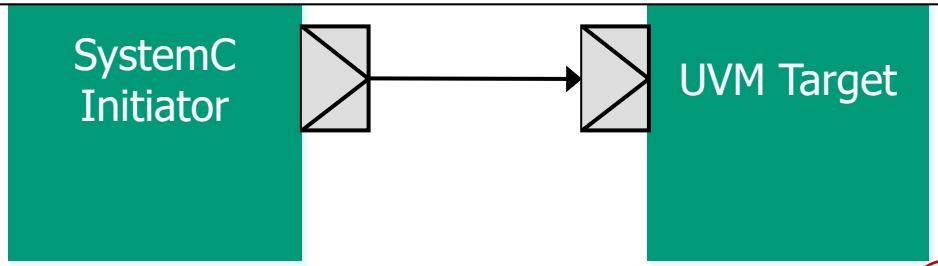
SystemVerilog UVM

```
function void connect_phase(uvm_phase phase);
  uvmc_tlm #()::connect(target1.in, "socket");
endfunction
```

Synchronization with UVM Connect

SystemC

```
uvmc_wait_for_phase("run",UVM_PHASE_STARTED);
uvmc_raise_objection("run");
...
sc_core::wait(run_done_ev);
uvmc_drop_objection("run");
```



SystemVerilog UVM

```
initial
  uvmc_init();
```

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

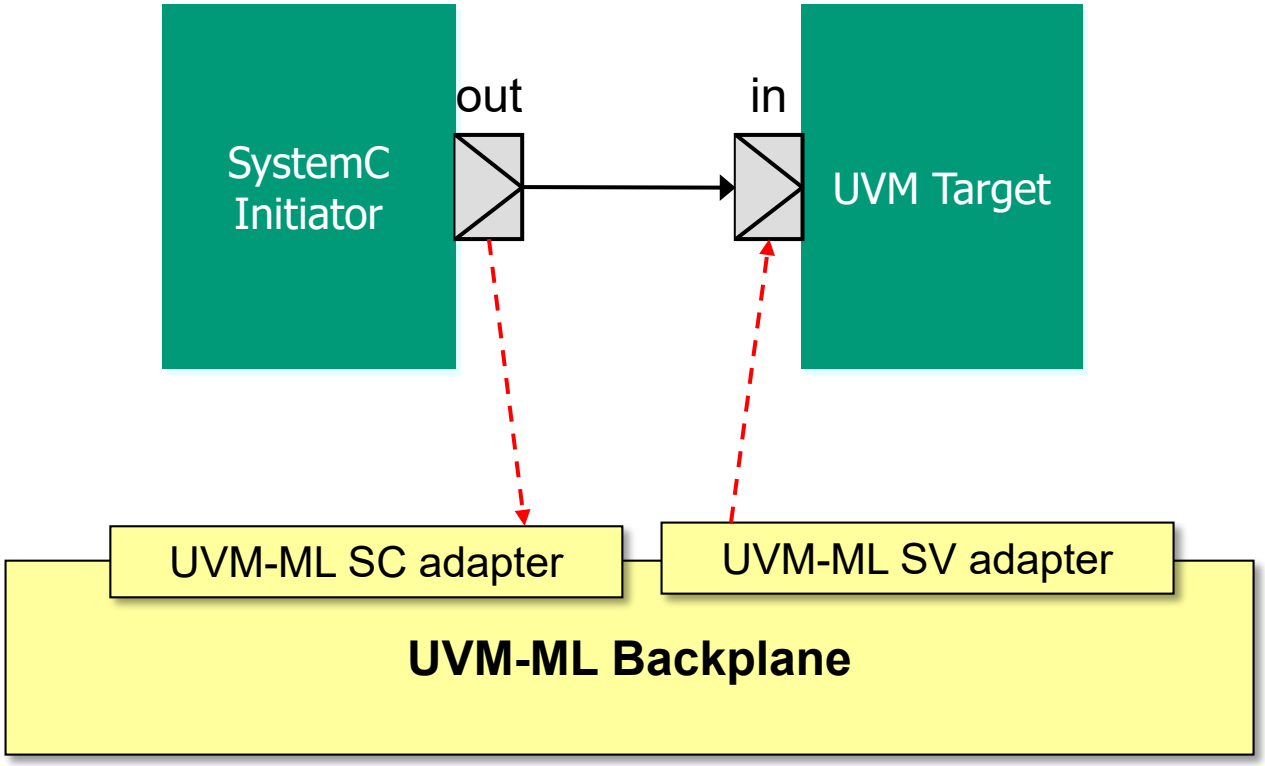
Agenda

- TLM in UVM and SystemC
- UVM Connect
- **UVM-ML**
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- Conclusions

UVM-ML

- Frameworks for SystemVerilog, SystemC and *e* verification environments connected via a multi-language backplane
- Provides SystemC versions of UVM base classes – SystemC components automatically synchronized to UVM phase scheduler
- Automatic and user-defined converters for transaction classes
- Based on standard SystemVerilog DPI-C so works with simulators from multiple vendors
- Developed by Cadence but released under Apache 2.0 License with all source code

TLM Connections with UVM-ML



UVM-ML Port/Export/Socket Register and Connect Functions

SystemC

```
ml_tlm2_register_initiator(this,initiator.out,"socket",
                           this->name());
```

SystemVerilog UVM

```
uvm_ml::ml_tlm2 #()::register(target1.in);
```

SystemVerilog UVM

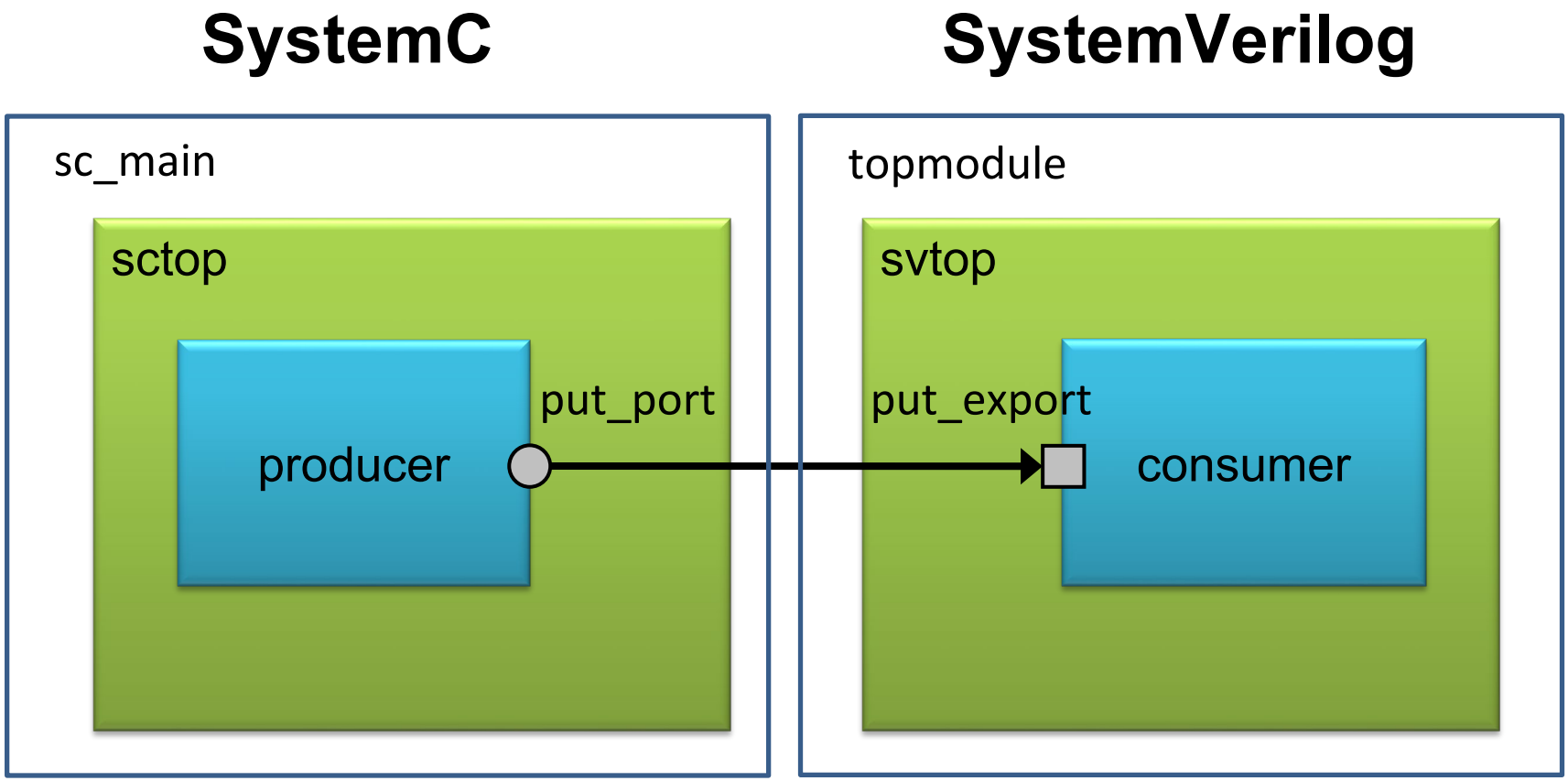
```
bit res = uvm_ml::connect("sys.prod.socket",
                          target1.in.get_full_name());
```

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- **TLM1 Producer-Consumer**
- TLM-2.0
- Future developments?
- Conclusions

TLM1 Example



UVM Connect SystemC Transaction Class

```
enum dir_t { READ, WRITE };

class trans_t {
public:
  int addr;
  std::vector<unsigned char> data;
  dir_t dir;

  trans_t() {
    data.reserve(8);  //allocate space for 8 chars
  }
  virtual void do_print(std::ostream& os) const;
};
```

UVM Connect SystemC Converter

- Template specialization of uvmc_converter class is required for user-defined transaction classes

```
template <>
struct uvmc_converter<trans_t> {
  static void do_pack(const trans_t &t,
                      uvmc::uvmc_packer &packer) {
    packer << t.addr << t.data << t.dir;
  }
  static void do_unpack(trans_t &t,
                        uvmc::uvmc_packer &packer) {
    packer >> t.addr >> t.data >> t.dir;
  }
};
```

UVM Connect SystemC Producer

```
class producer : public sc_module {
public:
  sc_port<tlm::tlm_blocking_put_if<trans_t> > put_port;
```

```
void run() {
  trans_t * t;
  sc_core::wait(10, SC_NS);
  ...
  put_port->put(*t);  ...
```

```
void run_proc() {
  uvmc_raise_objection("run");
  run();
  uvmc_drop_objection("run");
}
```

SystemC SC_THREAD

UVM Connect SystemVerilog Transaction Class

```
class trans_t extends uvm_sequence_item;
  `uvm_object_utils(trans_t)
  typedef enum {READ,WRITE} dir_t;
  rand int addr;
  rand byte data[$];
  rand dir_t dir;

  function new(string name=""); ...
  function void do_pack(uvm_packer packer); ...
  function void do_unpack(uvm_packer packer); ...

endclass: trans_t
```

Standard UVM code!

```
function void do_pack(uvm_packer packer);
  `uvm_pack_int(addr)
  `uvm_pack_queue(data)
  `uvm_pack_enum(dir)
endfunction: do_pack
```

UVM Connect SystemVerilog Consumer

```
class consumer extends uvm_component;
  uvm_blocking_put_imp #(trans_t,consumer) put_export;
  `uvm_component_utils(consumer)

  function new(string name, uvm_component parent=null);
    super.new(name,parent);
    put_export = new("put_export",  this);
  endfunction

  task put (input trans_t p);
    `uvm_info(get_type_name(), ... ,UVM_LOW);
    #1ns;
  endtask

endclass
```

Standard UVM code!

UVM-ML SystemC Transaction Class

```
#include "uvm.h"
enum dir_t { READ, WRITE };

class trans_t : public uvm::uvm_object {
public:
  UVM_OBJECT_UTILS(trans_t)
  ... //attributes and constructor same as the UVMC version

  virtual void do_pack(uvm::uvm_packer &packer) const;
  virtual void do_unpack(uvm::uvm_packer &packer);
  virtual void do_print(std::ostream& os) const;
  virtual void do_copy(const uvm::uvm_object* rhs);
  virtual bool do_compare(const uvm::uvm_object* rhs) const;
};
```

Virtual functions must be overridden – pure virtual in uvm_object!

```
UVM_OBJECT_REGISTER(trans_t)
```

Register class with backplane

UVM-ML SystemC Producer

```
class producer : public uvm::uvm_component {
public:
  sc_port<tlm::tlm_blocking_put_if<trans_t> > put_port;

  producer(sc_core::sc_module_name nm) :
     uvm_component(nm), put_port("put_port"){
    // nothing to do in constructor!
    }
  UVM_COMPONENT_UTILS(producer)
```

```
void run() {
  trans_t * t;
  sc_core::wait(10, SC_NS);
  ...
  put_port->put(*t);  ...
```

SystemC SC_THREAD

Synchronized with UVM run_phase

UVM-ML SystemC Top

```
class sctop : public sc_core::sc_module {
public:
  producer prod;
  sctop(sc_core::sc_module_name nm) : sc_module(nm)
                          , prod("prod") {
    uvm_ml::uvm_ml_register(&prod.put_port);
  }
  void before_end_of_elaboration() {
    uvm_ml::uvm_ml_connect(prod.put_port.name(),
                           "svtop.consumer.put_export");
  }
};
```

Alternatively, can call connect function from SV

```
int sc_main(int argc, char** argv) {
  return 0;
}
```

Note empty sc_main

```
UVM_ML_MODULE_EXPORT(sctop)
UVM_COMPONENT_REGISTER(producer)
```

UVM-ML SystemVerilog Top

- Transaction and Consumer classes same as UVMC example

```
class svtop extends uvm_env;
  consumer cons;
  function void phase_ended(uvm_phase phase);
    if (phase.get_name() == "build")
      uvm_ml::ml_tlm1#(trans_t)::register(cons.put_export);
    end
  endfunction
```

```
module topmodule;
initial begin
  string tops[2];
  tops[0] = "SC:sctop";
  tops[1] = "SV:svtop";
  uvm_ml_run_test(tops, "");
end
```

Table contains names of top-level modules

Note: Not UVM run_test()

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- **TLM-2.0**
- Future developments?
- Conclusions

TLM-2.0 Blocking Transport

- UVM Connect and UVM-ML both support TLM-2.0
- Converting TLM1 producer-consumer to blocking transport is trivial
  - port/export replaced by initiator/target socket
  - transaction class replaced by generic_payload
  - do_pack/do_unpack for generic_payload built-in
  - TLM1 connect/register functions replaced by TLM-2.0 versions
- Transaction object automatically copied between SC and SV at transaction start and end

TLM-2.0 Non-Blocking Transport

- Transaction lifetime extends over multiple phases of "base protocol" (multiple calls to nb_transport)
- Standard requires a memory manager (not provided by UVM)
  - No memory manager in UVM Connect. SV transactions garbage collected when phase reaches END_RESP
  - UVM-ML memory manager requires full base protocol to be used (may get memory leaks with early completion)
- Complex scenarios (pipelined transactions, payload event queues) require custom code (and great care)

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- **Future developments?**
- Conclusions

Future Developments?

- UVM-SystemC
  - A complete implementation of UVM class hierarchy in SystemC
  - Should simplify integration of SC and SV UVM
  - Still need a mechanism to synchronize phases and pass transactions

- Multi-Language Working Group
  - A standard, open-source architecture to integrate SystemVerilog and SystemC UVM?

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- TLM in UVM and SystemC
- UVM Connect
- UVM-ML
- TLM1 Producer-Consumer
- TLM-2.0
- Future developments?
- **Conclusions**

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Conclusions

- UVM Connect and UVM-ML both provide portable solutions for integration of SystemC and UVM TLM models
  - It is no longer necessary to create your own solution!
- UVM Connect
  - Easiest to learn and create first examples
- UVM-ML
  - More powerful and closer integration with UVM
  - More support for TLM-2.0 models

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Further Information

- Additional details given in the paper
- Source code: see Doulos website after DVCon

www.doulos.com/knowhow

- Any Questions?