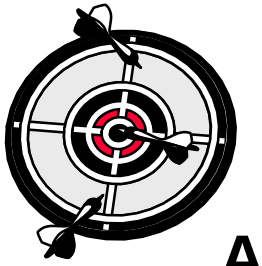
# Agenda: Day 2

**DAY 2**

| | |
|---|---|
| 5 | UVM Configuration & Factory |
| 6 | UVM Component Communication  |
| 7 | UVM Scoreboard & Coverage |
| 8 | UVM Callback  |

# Unit Objectives

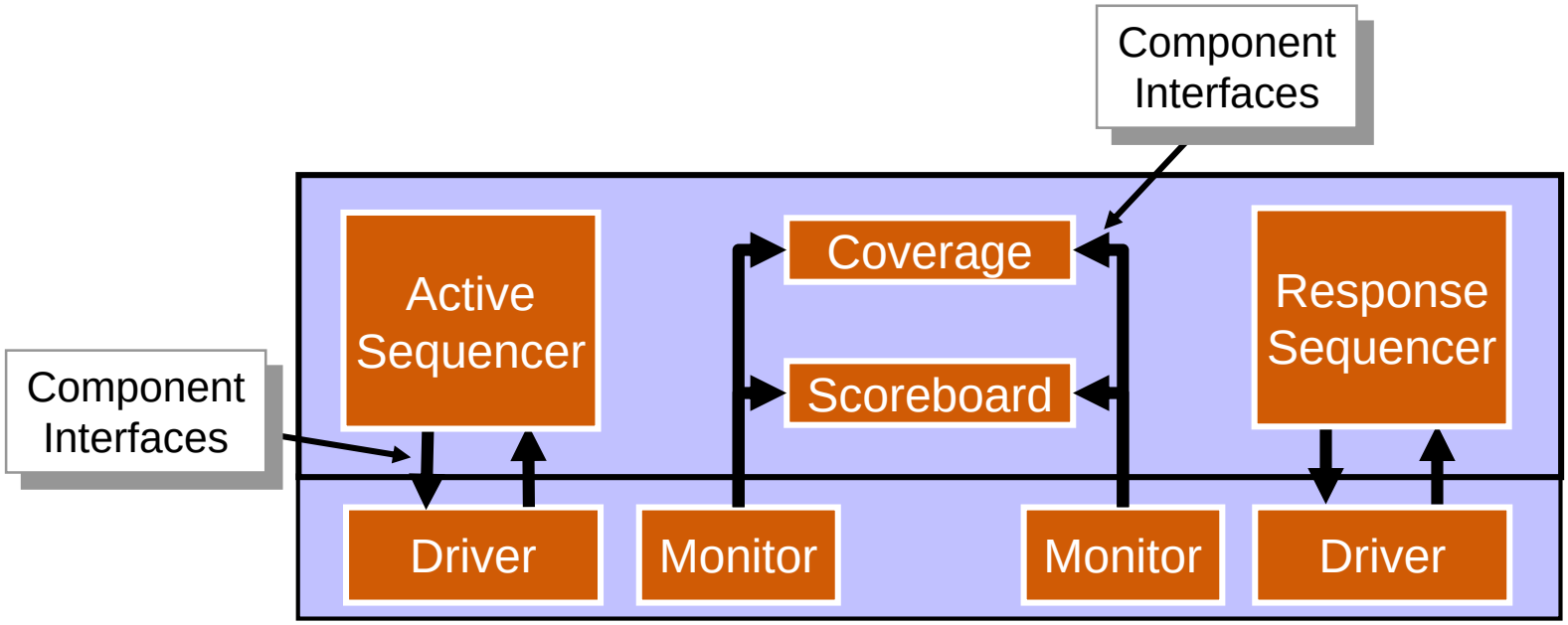


**After completing this unit, you should be able to:**

- **Describe and implement TLM port/socket for communication between components**

# Component Communication: Overview

- **Need to exchange transactions between components of verification environment**
  - Sequencer → Driver
  - Monitor → Collectors (Scoreboard, Coverage)

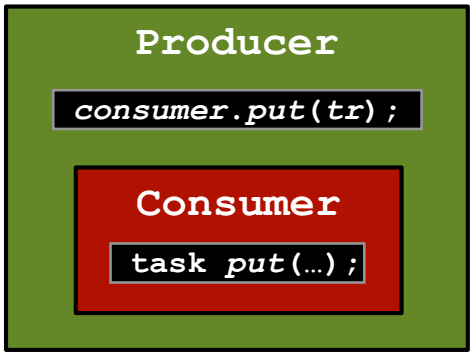


# Component Communication: Method Based

- **Component can embed method for communication**

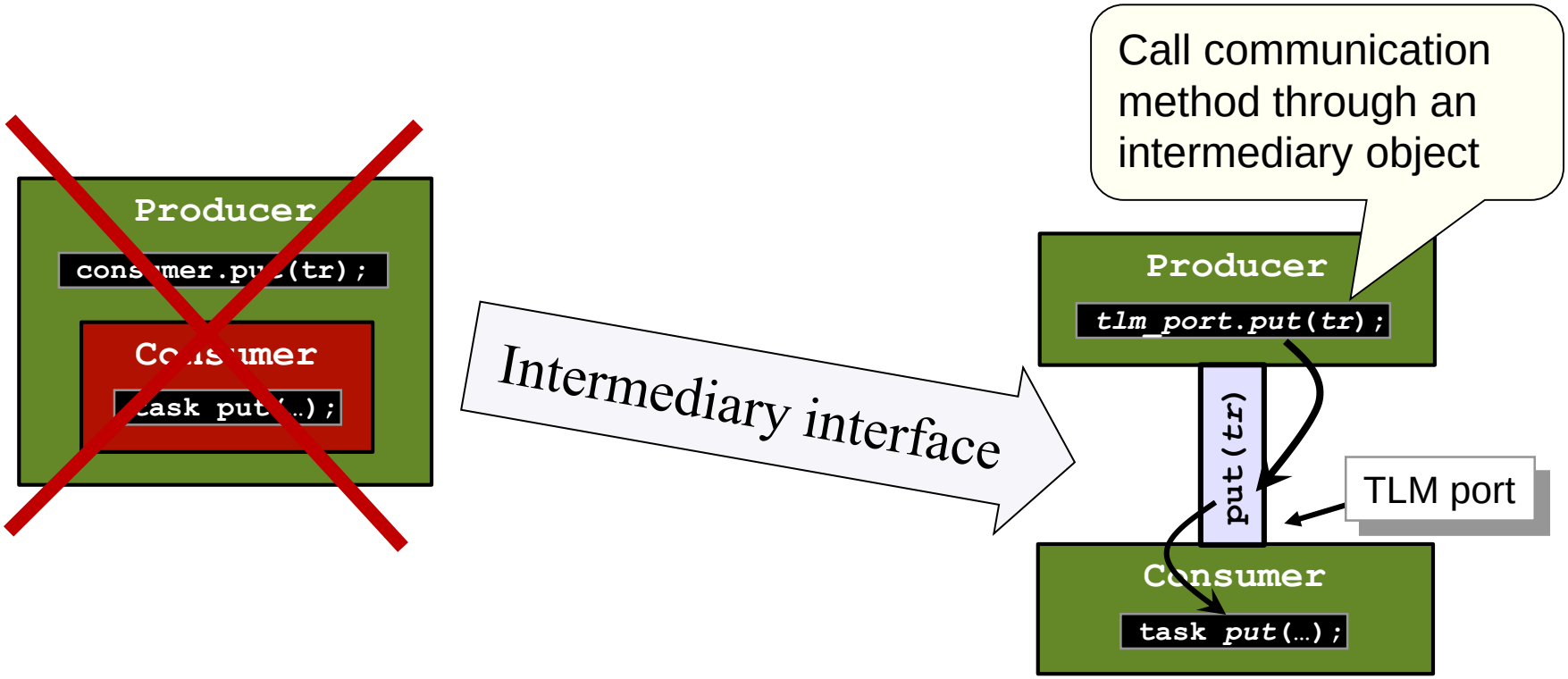
```
class Consumer extends uvm_component;
  ...
  virtual task put(transaction tr);
endclass
```

- **But, the communication method should not be called through the component object's handle**
  - Code becomes too inflexible for testbench structure
    - In example below, Producer is stuck with communicating with only a specific Consumer type

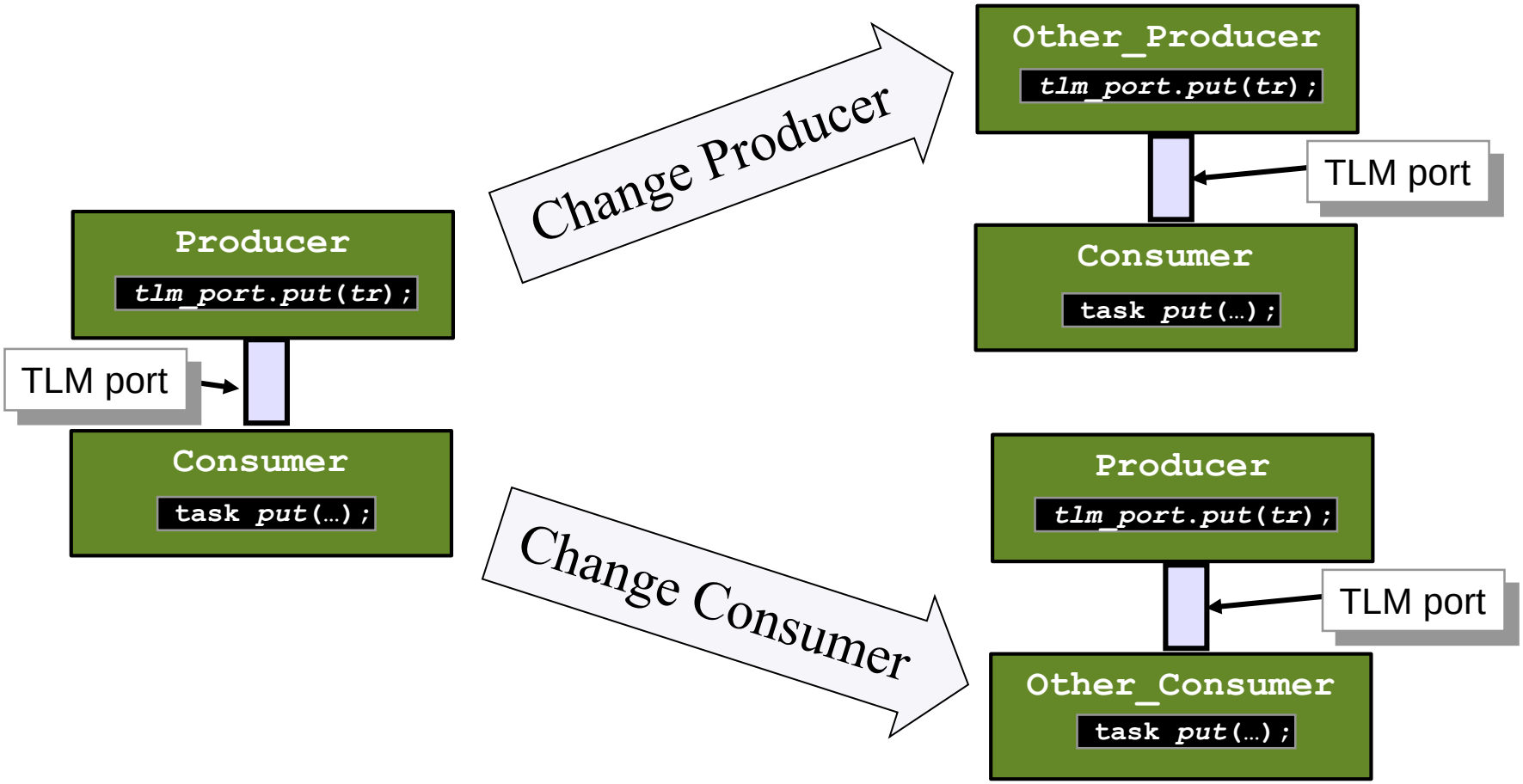
# Component Communication: TLM

- **Use an intermediary object (TLM) to handle the execution of the procedural communication**



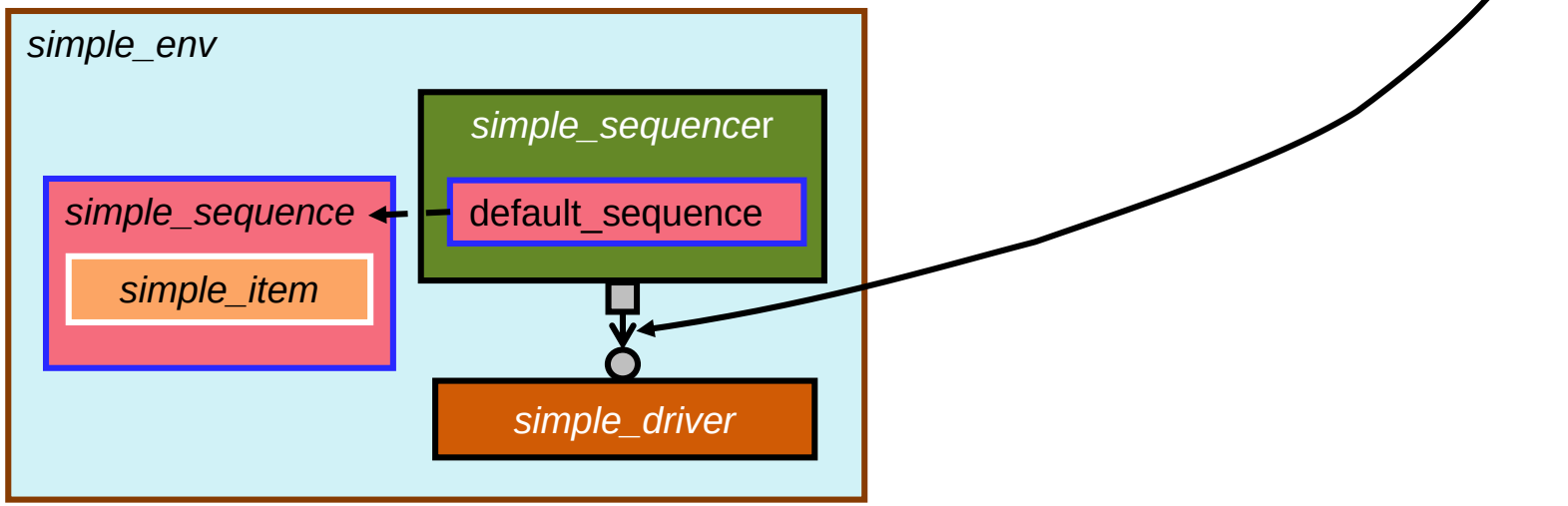
# Component Communication: TLM

- **Through the intermediary object (TLM) components can be re-connected to any other component on a testcase by testcase basis**

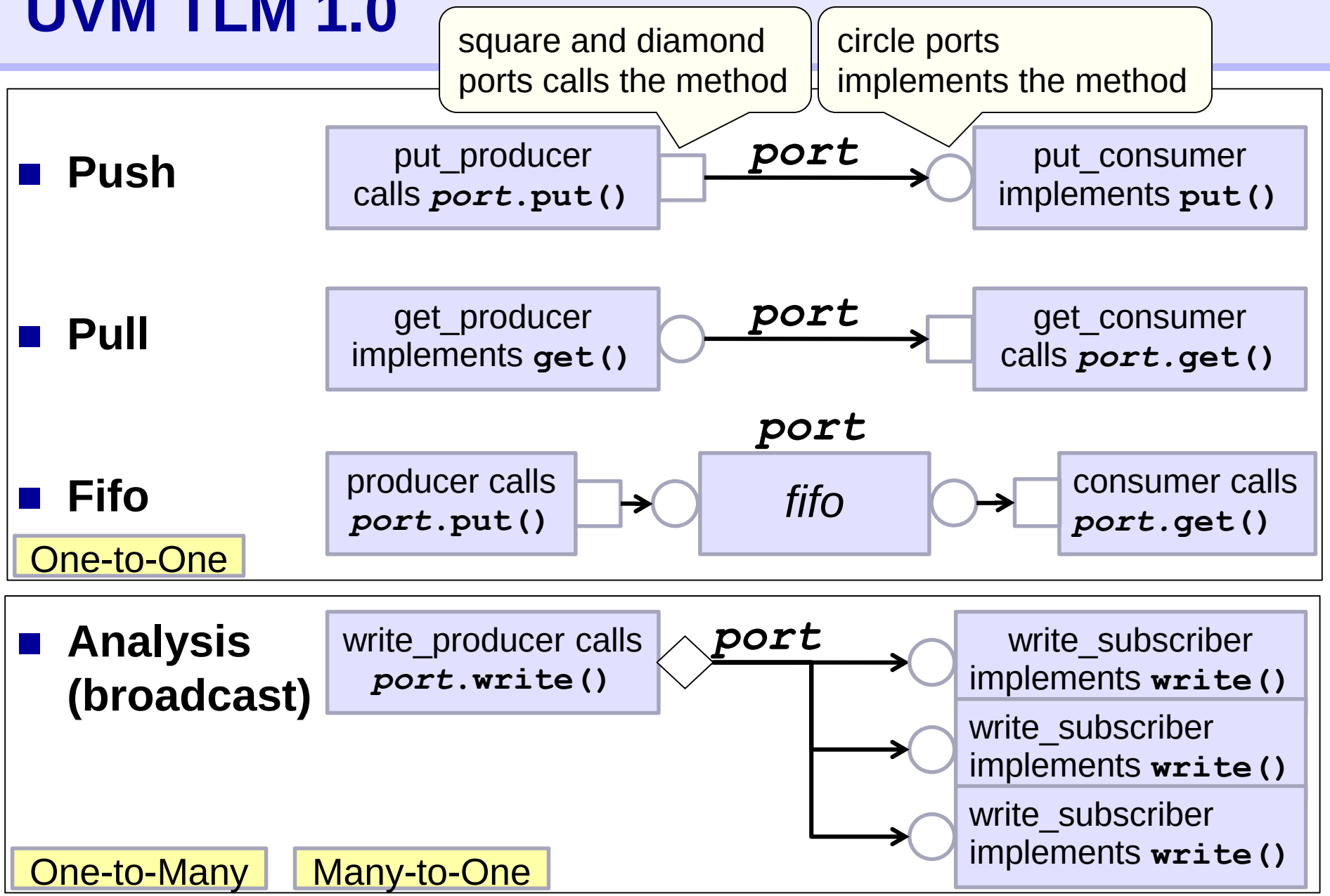


# Communication in UVM: TLM 1.0, 2.0

- TLM Classes
  - `uvm_*_export`
  - `uvm_*_imp`
  - `uvm_*_port`
  - `uvm_*_fifo`
  - `uvm_*_socket`



# UVM TLM 1.0


square and diamond ports calls the method
circle ports implements the method
Push
put_producer calls port.put()
port
put_consumer implements put()
Pull
get_producer implements get()
port
get_consumer calls port.get()
Fifo
port
producer calls port.put()
fifo
consumer calls port.get()
One-to-One
Analysis (broadcast)
write_producer calls port.write()
port
write_subscriber implements write()
write_subscriber implements write()
write_subscriber implements write()
One-to-Many
Many-to-One

# Push Mode

- **Push mode**


```
class producer extends uvm_component; ...
  uvm_blocking_put_port #(packet) put_port;
  function void build_phase(uvm_phase phase); ...
    put_port = new("put_port", this);
  endfunction
  virtual task initiate_tr(); ...
    put_port.put(tr);
  e
end
```

```
class consumer extends uvm_component; ...
  uvm_blocking_put_imp #(packet, consumer) put_export;
  function void build_phase(uvm_phase phase); ...
    put_export = new("put_export", this);
  endfunction
  virtual task put(packet tr);
    process_tr(tr);
```

```
class environment extends uvm_env; producer p; consumer c; ...
  virtual function void connect_phase(uvm_phase phase); ...
    p.put_port.connect(c.put_export);  // connection required!
  endfunction
endclass
```

# Pull Mode

- **Pull mode**

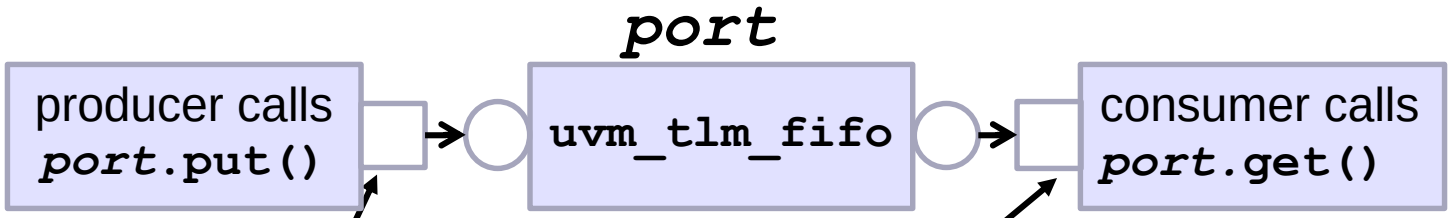

```
class producer extends uvm_component; ...
  uvm_blocking_get_imp #(packet, producer) get_export;
  virtual task get(output packet tr);
    tr = packet::type_id::create("tr", this); ...
  endtask
endclass
```

```
class consumer extends uvm_component; ...
  uvm_blocking_get_port #(packet) get_port;
  virtual task retrieve_tr(); ...
    get_port.get(tr);
  endtask
endclass
```

```
class environment extends uvm_env; ...
  producer p;  consumer c;
  virtual function void connect_phase(uvm_phase phase); ...
    c.get_port.connect(p.get_export); // connection required!
  endfunction
endclass
```

# FIFO Mode

- **FIFO Mode**

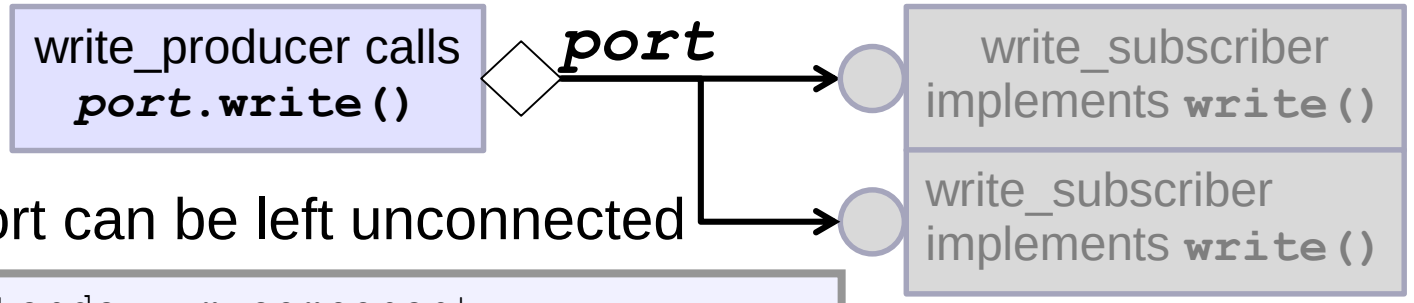



- **Connect producer to consumer via `uvm_tlm_fifo`**

```
class environment extends uvm_env; ...
  producer p;
  consumer c;
  uvm_tlm_fifo #(packet) tr_fifo;
  virtual funtion void build_phase(uvm_phase phase); ...
    p = producer::type_id::create("p", this);
    c = consumer::type_id::create("c", this);
    tr_fifo = new("tr_fifo", this); // No proxy (type_id) for TLM ports
  endfunction
  virtual function void connect_phase(uvm_phase phase); ...
    p.put_port.connect(tr_fifo.put_export); // connection required!
    c.get_port.connect(tr_fifo.get_export); // connection required!
  endfunction
endclass
```

# Analysis Port

- **Analysis (broadcast)**
  - Analysis port can be left unconnected



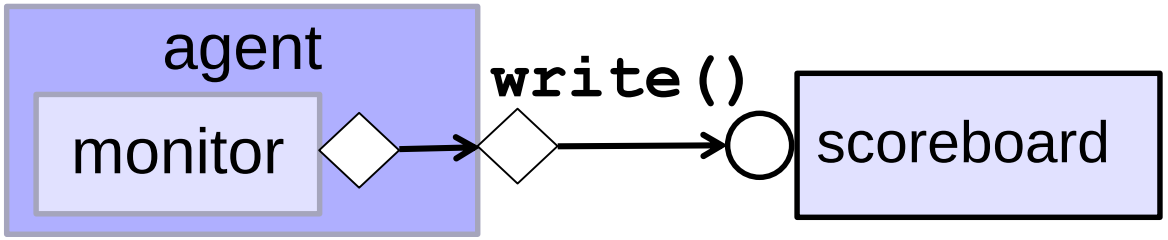

```
class producer extends uvm_component; ...
  uvm_analysis_port #(packet) analysis_port;
  virtual task assemble_tr(); ...
    analysis_port.write(tr);
  endtas
endclass
```

```
class subscriber extends uvm_component; ...
  uvm_analysis_imp #(packet, subscriber) analysis_export;
  virtual function void write(packet tr); // cannot block
    process_transaction(tr);
  endfunction
```

```
class environment extends uvm_env; ...
  producer p; subscriber s0, s1;  // other subscribers
  virtual function void connect_phase(uvm_phase phase); ...
    p.analysis_port.connect(s0.analysis_export);
    p.analysis_port.connect(s1.analysis_export);
  endfunction
endclass
```

# Port Pass-Through

- **Connecting sub-component TLM ports**
  - Use same port type



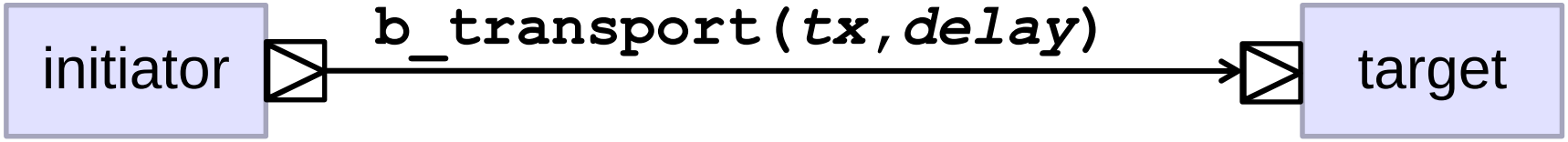

```
class monitor extends uvm_monitor; // other code not shown ...
  uvm_analysis_port #(packet) analysis_port;
  virtual function void build_phase(uvm_phase phase); ...
    this.analysis_port = new("analysis_port", this);
  endfunction
endclass
```

```
class agent extends uvm_agent; // other code not shown ...
  monitor mon;
  uvm_analysis_port #(packet) analysis_port;
  virtual function void build_phase(uvm_phase phase); ...
    this.analysis_port = new("analysis_port", this);
  endfunction
  virtual function void connect_phase(uvm_phase phase); ...
    mon.analysis_port.connect(this.analysis_port);
  endfunction
endclass
```

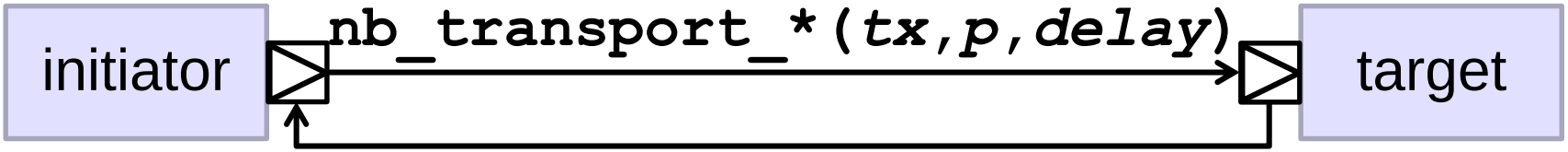
port must be same type

# UVM TLM 2.0

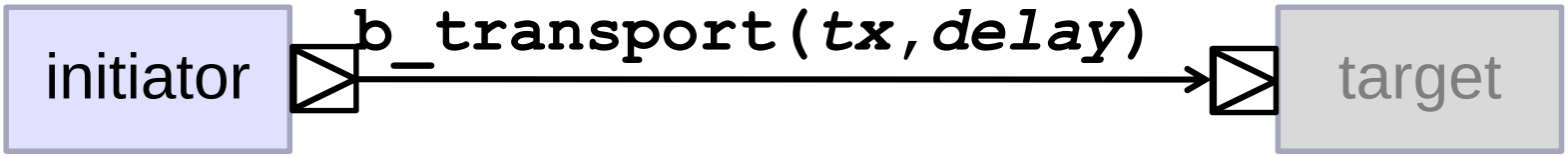
- **Blocking**




- **Non-Blocking**

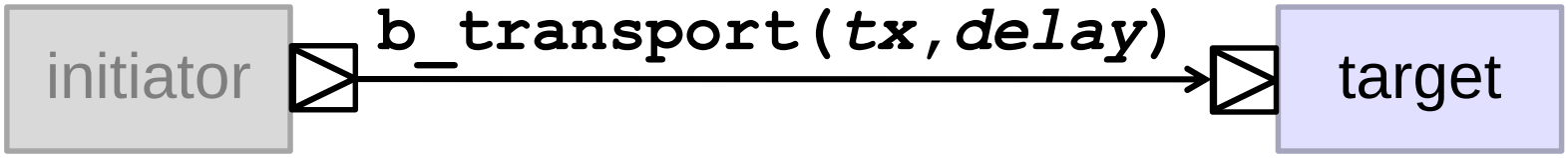


# Blocking Transport Initiator




```
class initiator extends uvm_component;
  uvm_tlm_b_initiator_socket #(packet) i_socket;
  // constructor not shown
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    i_socket=new("i_socket", this);
  endfunction
  virtual task initiate_tr();
    packet tx = packet::type_id::create("tx", this);
    uvm_tlm_time delay = new();
    delay.set_abstime(1.5, 1e-9);   // set delay to 1.5ns
    i_socket.b_transport(tx, delay);
  endtask
endclass
```

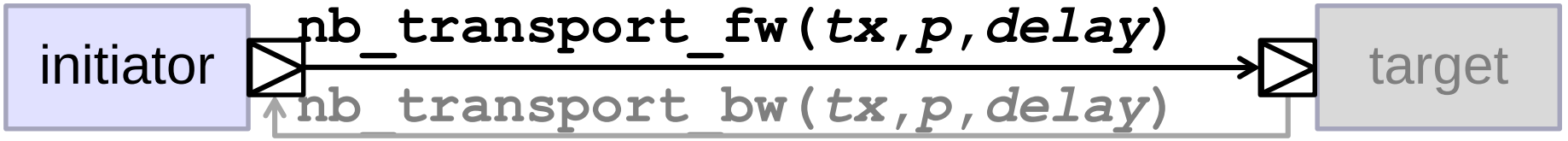
# Blocking Transport Target




```
class target extends uvm_component; ...
  uvm_tlm_b_target_socket #(target, packet) t_socket;
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    t_socket=new("t_socket", this);
  endfunction
  virtual task b_transport(packet tx, uvm_tlm_time delay);
    $display("realtime = %t", delay.get_realtime(1ns));
    ...
  endtask
endclass
```

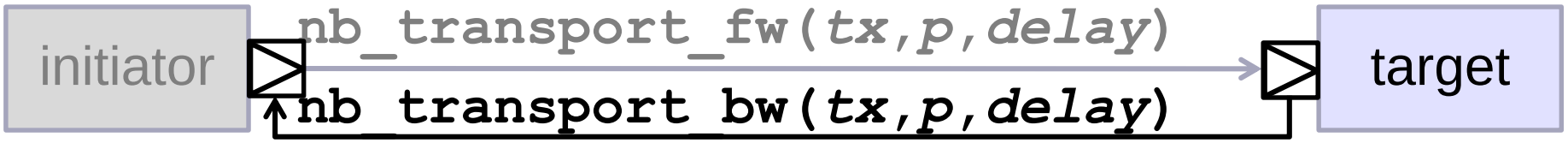
```
class environment extends uvm_env;
  initiator intr;
  target    trgt;
  // component_utils, constructor and build_phase not shown
  virtual function void connect_phase(uvm_phase phase);
    intr.i_socket.connect(trgt.t_socket);
  endfunction
endclass
```

# Non-Blocking Transport Initiator




```
class initiator extends uvm_component;
  uvm_tlm_nb_initiator_socket #(initiator, packet) i_socket;
  // component_utils, constructor and build_phase not shown
  virtual task initiate_tr();
    uvm_tlm_sync_e sync; uvm_tlm_phase_e p; uvm_tlm_time delay = new;
    packet tx = packet::type_id::create("tx", this);
    if (!tx.randomize()) begin
      `uvm_fatal("RAND_ERR", "tx randomize failed");
    end
    sync = i_socket.nb_transport_fw(tx, p, delay);
  endtask
  virtual function uvm_tlm_sync_e nb_transport_bw(packet tx,
                    ref uvm_tlm_phase_e p, input uvm_tlm_time delay);
    // ... Process acknowledgement from target
    return (UVM_TLM_COMPLETED);
  endfunction
endclass
```

# Non-Blocking Transport Target




```
class target extends uvm_component;
  uvm_tlm_nb_target_socket #(target, packet) t_socket;
  // component_utils, constructor and build_phase not shown
  virtual function uvm_tlm_sync_e nb_transport_fw(packet tx,
                      ref uvm_tlm_phase_e p, input uvm_tlm_time delay);
    tx.print();
    fork  process_tr(tx);  join_none  // for delayed acknowledgement
    return (UVM_TLM_ACCEPTED);
  endfunction
  virtual task process_tr(packet tx);
    uvm_tlm_sync_e sync; uvm_tlm_phase_e p; uvm_tlm_time delay = new;
    // ... After completion of tx processing
    sync = t_socket.nb_transport_bw(tx, p, delay);
  endtask
endclass
```

# Unit Objectives Review

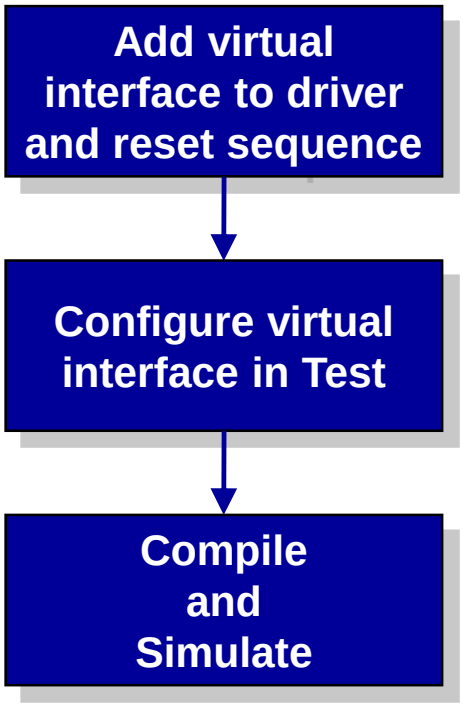
**Having completed this unit, you should be able to:**

- **Describe and implement TLM port/socket for communication between components**

# Lab 3 Introduction

60 min

## Implement & configure physical device drivers



# Appendix

**TLM 2.0 Generic Payload**

**DVE UVM-Aware Debugging Features**

**Verdi UVM-Aware Debugging Features**

# Verdi UVM-Aware Debugging Features

# Verdi UVM Debug Switches

**For VCS 2016.06 onwards**

- **Compile-time switches:**
  - **Requires `-debug_access+all`**
- **Post Simulation run-time switches:**

```
simv +UVM_VERDI_TRACE +UVM_TR_RECORD +UVM_LOG_RECORD \
     +UVM_TESTNAME=test_base

verdi -ssf novas.fsdb -nologo &
```

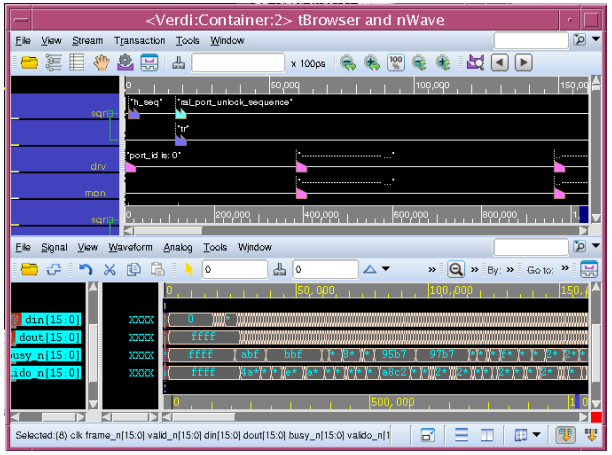
- **Interactive Simulation run-time switches**

```
simv +UVM_VERDI_TRACE +UVM_TR_RECORD +UVM_LOG_RECORD \
     +UVM_TESTNAME=test_base -gui=verdi
```

# UVM-Aware Features in Verdi

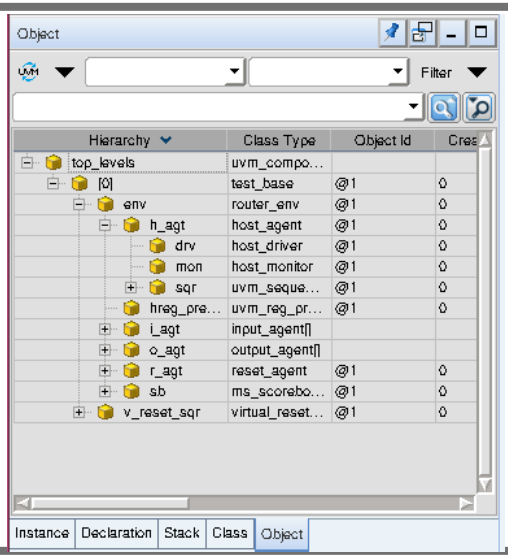
- **Post simulation debug**

## Transaction Recording

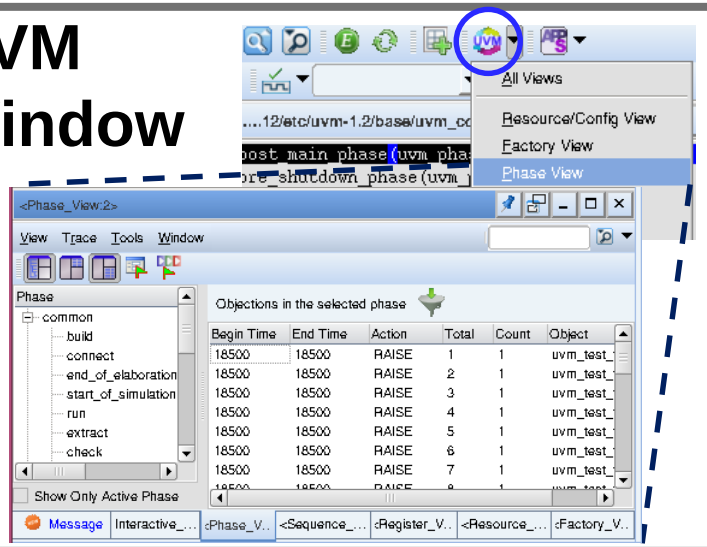



- **Interactive debug**

## Object Hierarchy Browser

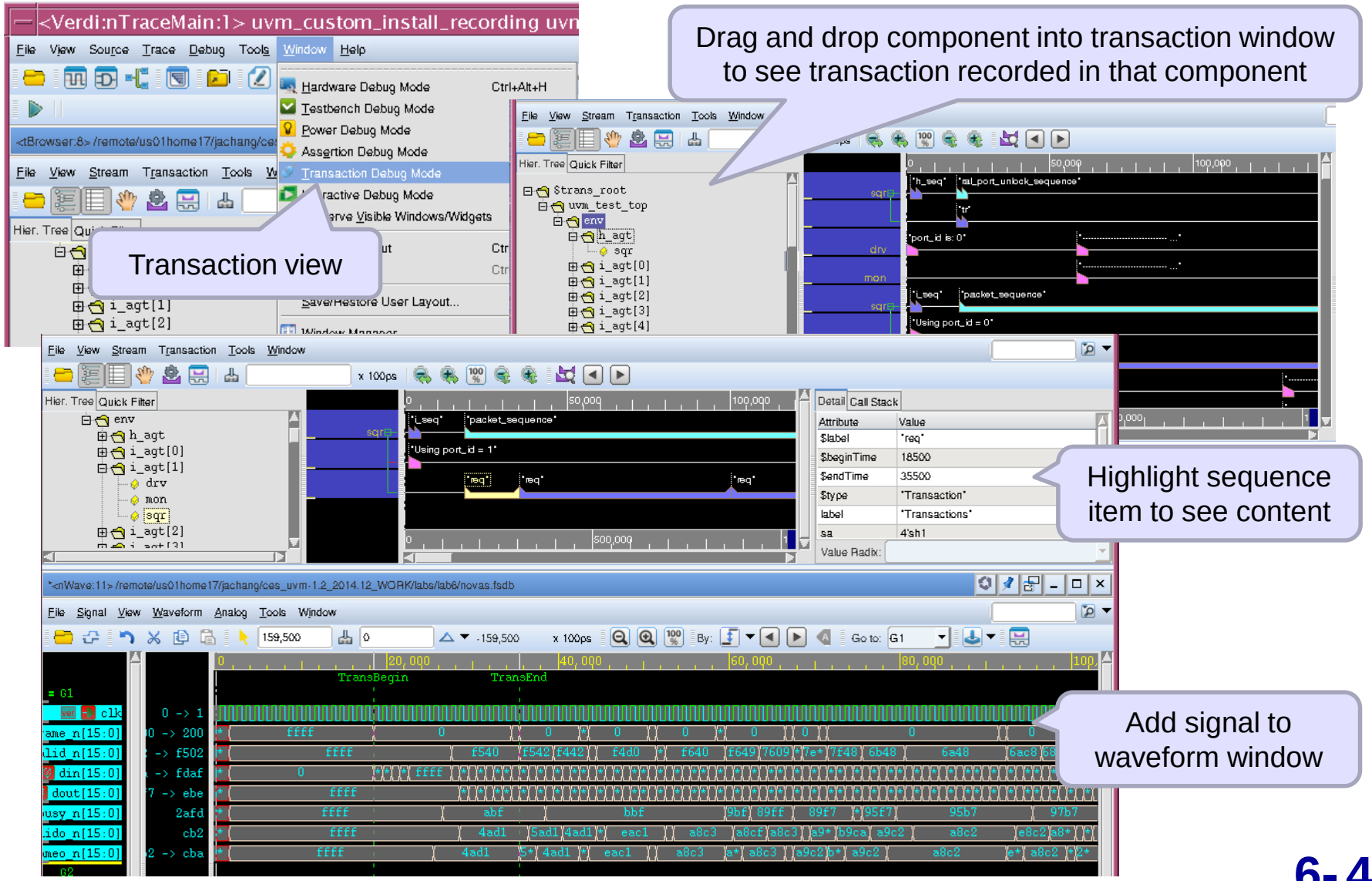



## UVM Window



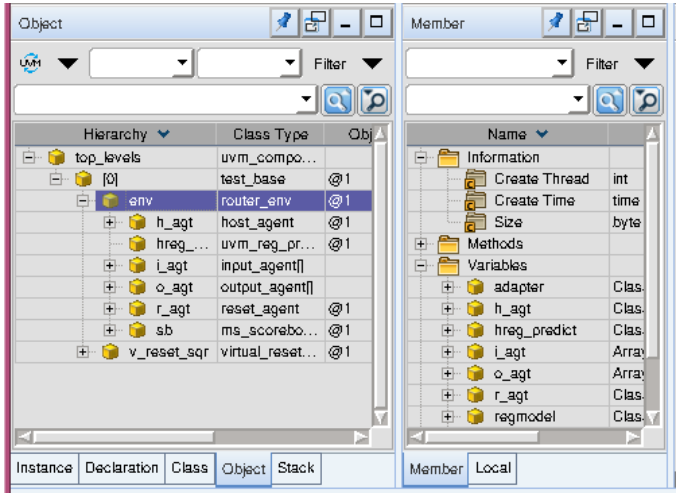
# UVM Transaction and Log Debug

- **Available for both post and interactive simulation**

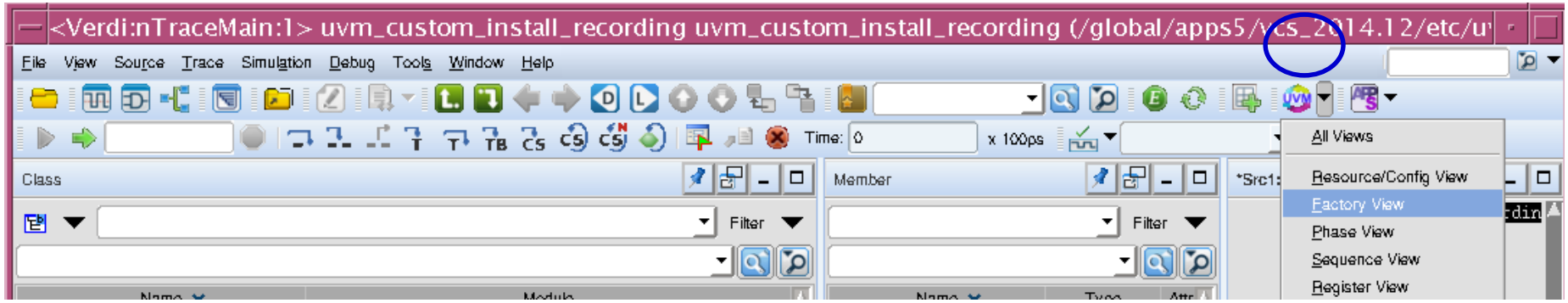


# Object Hierarchy Browser

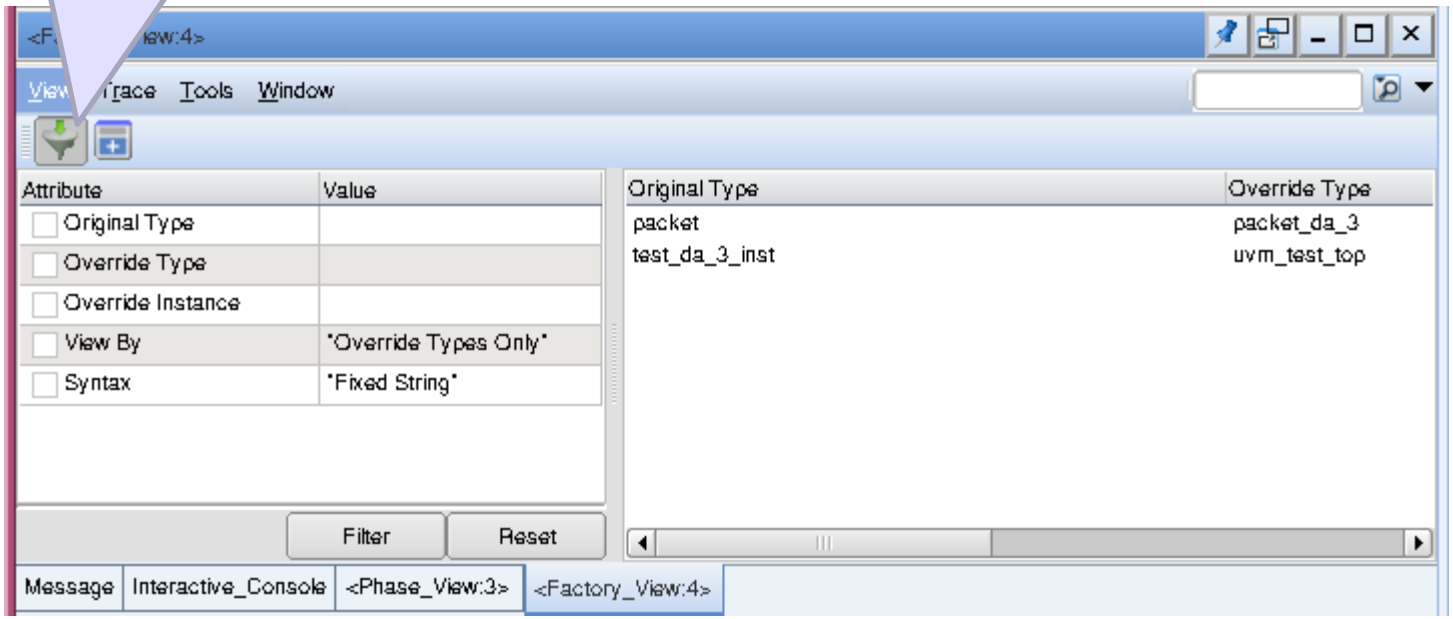
- **General mode and methodology-aware mode**
  - All Objects
  - UVM (Objects/Components)
  - OVM (Objects/Components)
  - VMM (Objects/Components)
- **Display and navigate all dynamic variables**
  - Testbench hierarchy view
  - Object creation time
  - Object thread ids
  - Object references
  - Dynamic memory profiling
  - Linked with Member Pane value annotation



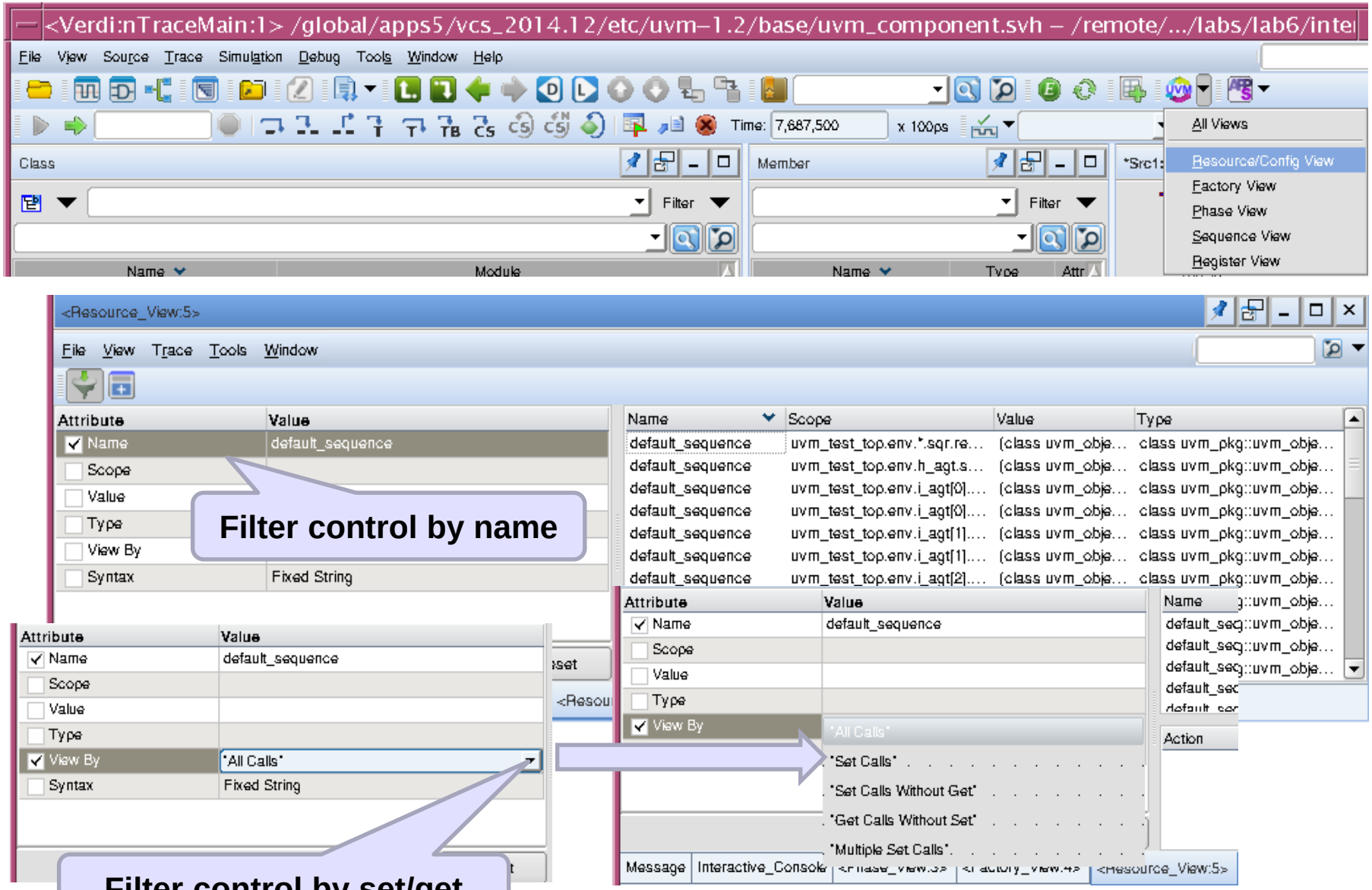
# UVM Factory Debug



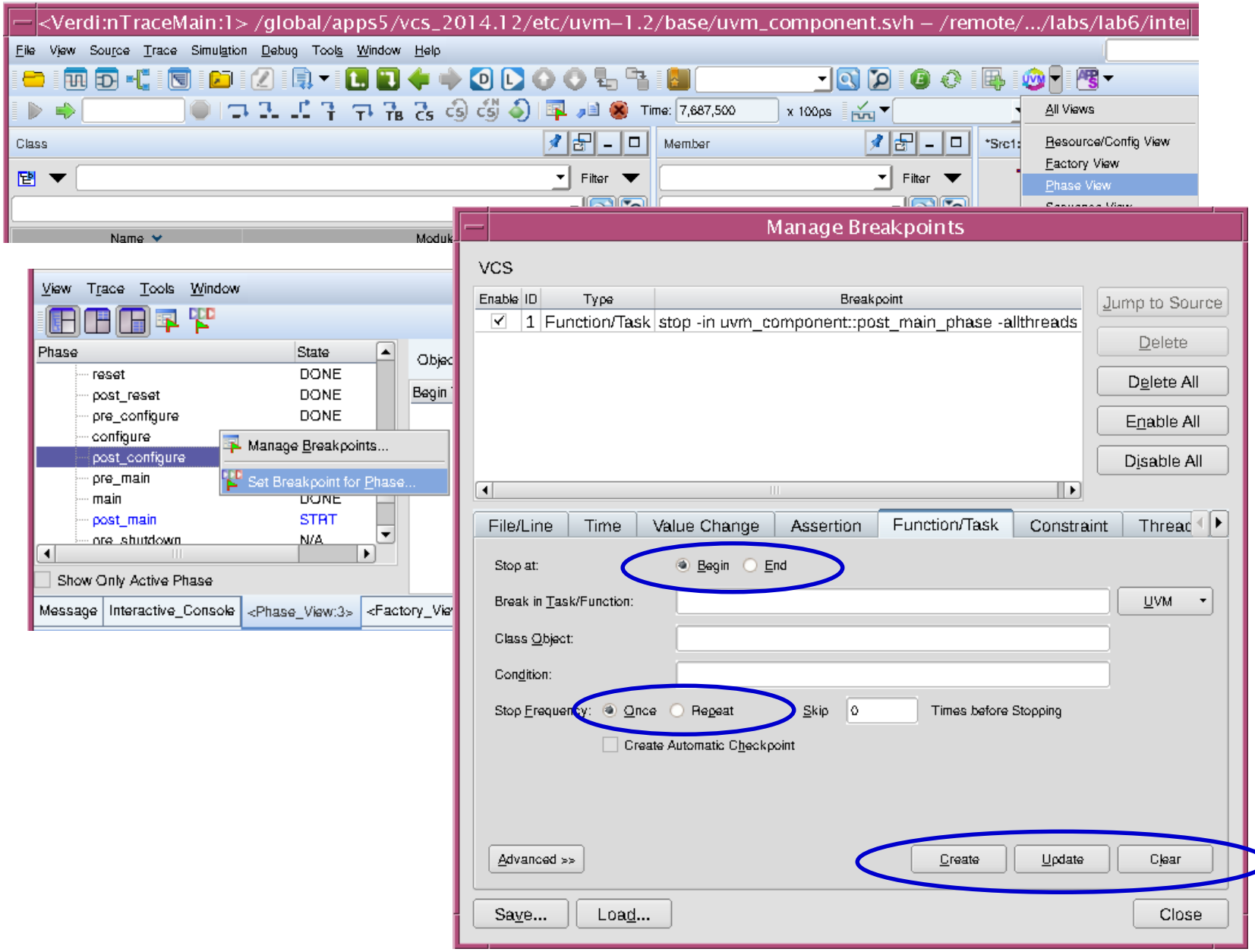

**Filter control by name**



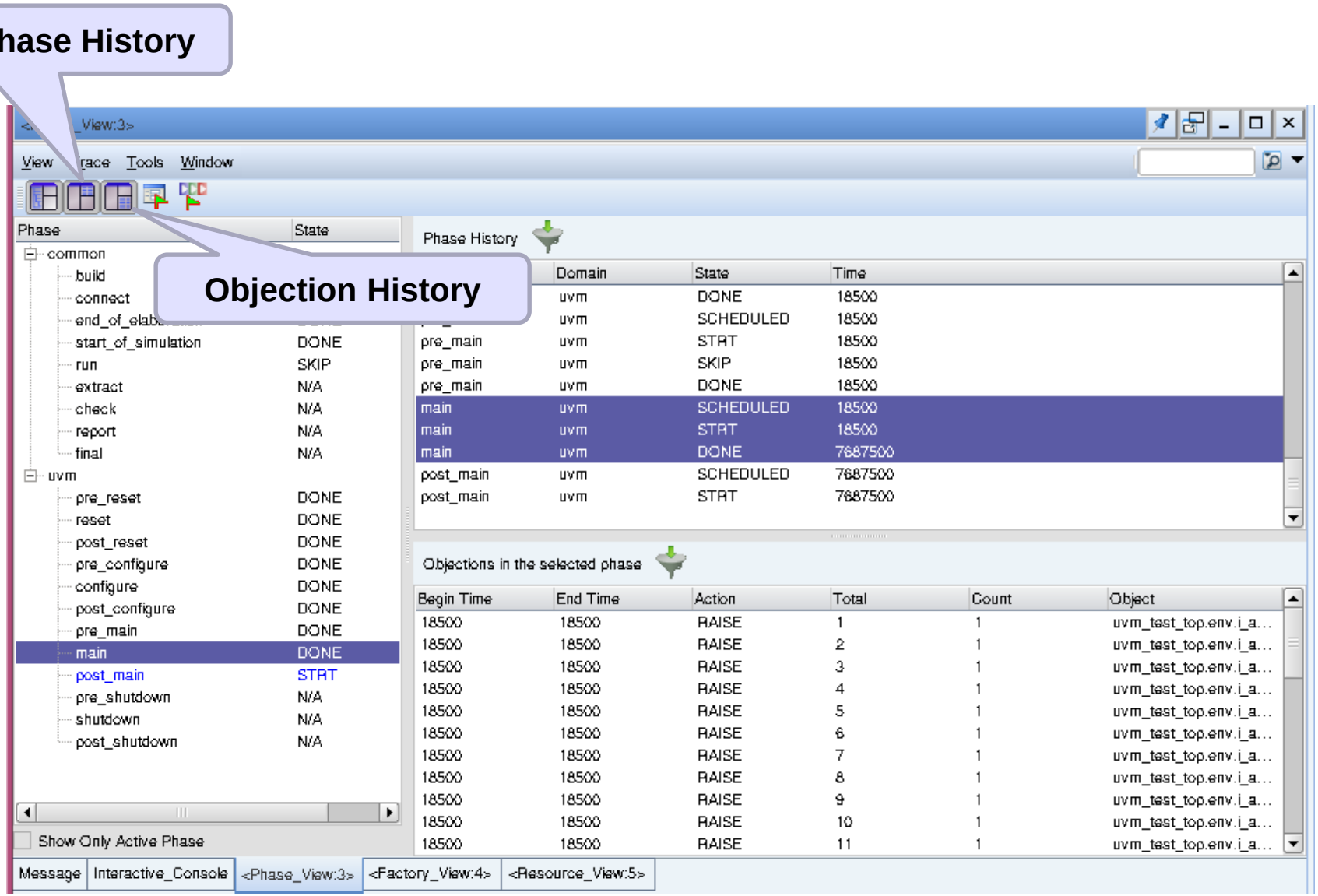
# UVM Resource Debug



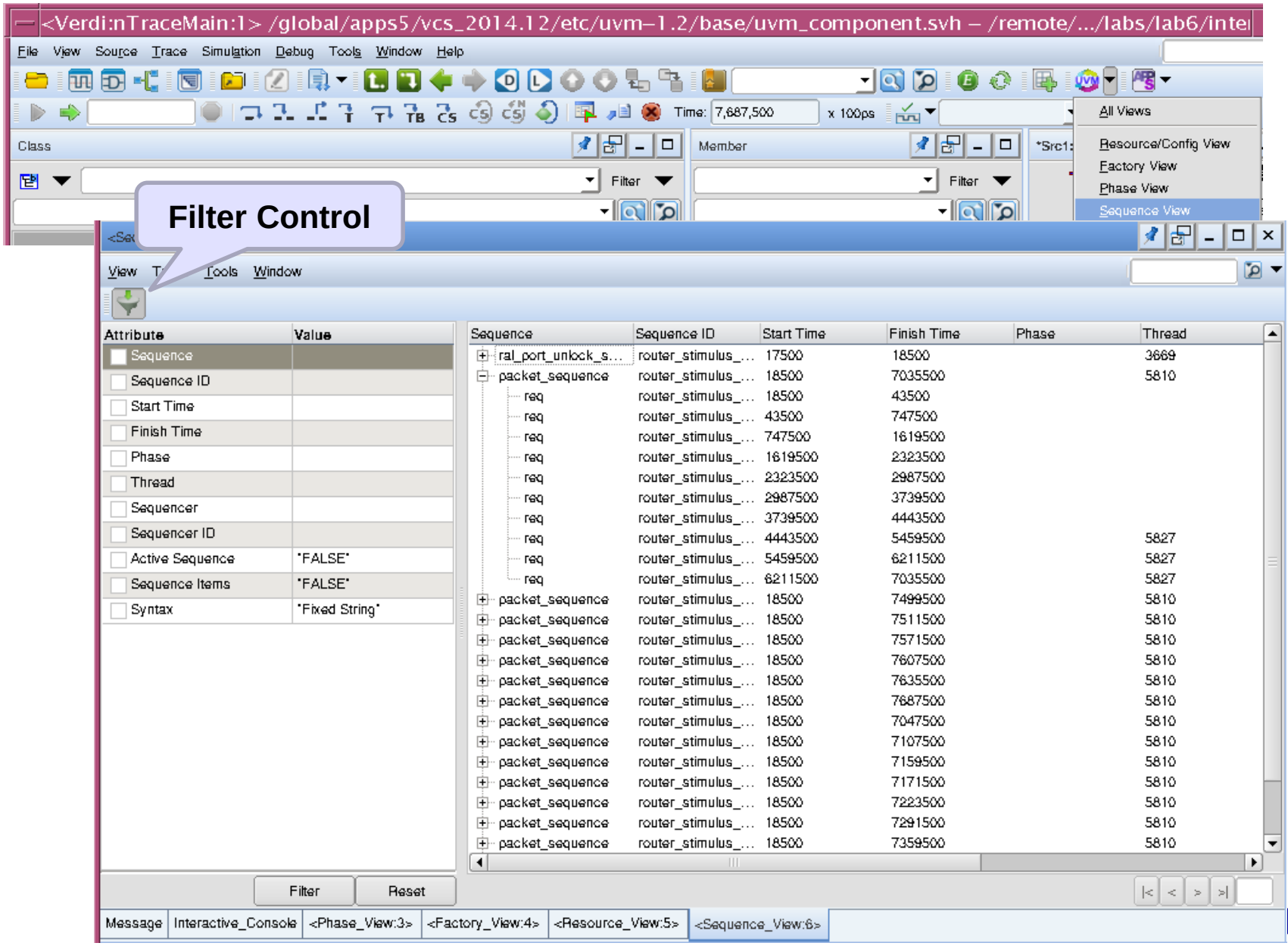
# UVM Phase-Based Breakpoint



# UVM Phase Objection Debug



# UVM Sequence Debug



# UVM Register Debug