Easier UVM – Making Verification Methodology More Productive

John Aynsley, Dr David Long, Doulos

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

What is UVM?

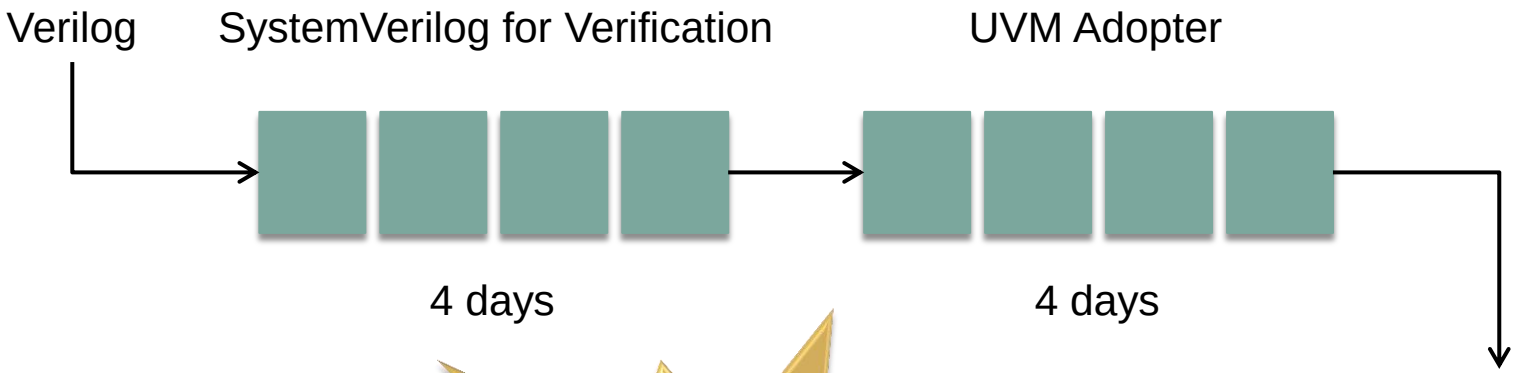
- The Universal Verification Methodology for SystemVerilog
- Supports constrained random, coverage-driven verification
- An open-source SystemVerilog base class library
- An Accellera standard!
- Supported by all major simulator vendors

Why UVM?

- **Best practice**
  - Consistency, uniformity, don't reinvent the wheel, avoid pitfalls
- **Reuse**
  - Verification IP, verification environments, tests, people, knowhow

UVM Itself is Challenging

- Doulos training



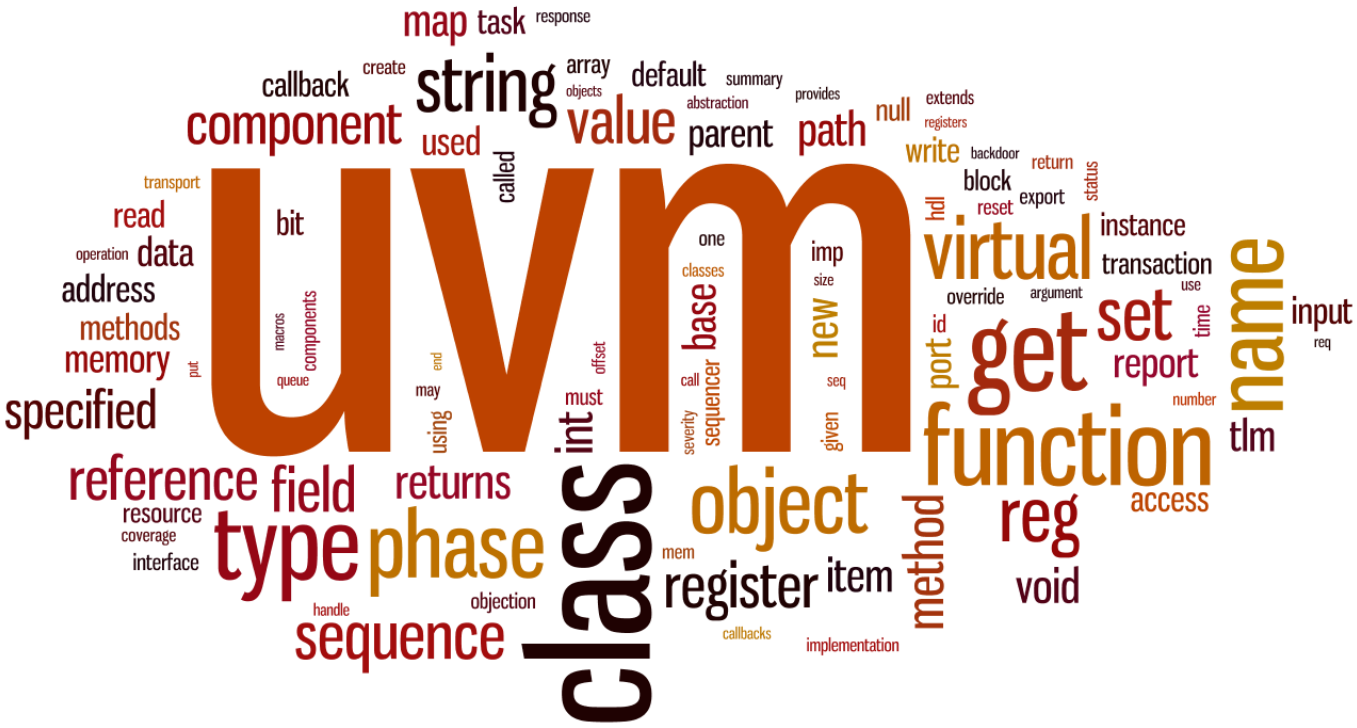
What next?

Tools?

Coaching?

Guidelines?

UVM Itself is Challenging



- ~ 300 classes in UVM Base Class Library
- The UVM documentation does not answer all the questions
- There are many ways to do the same thing - need a methodology

Easier UVM

- Coding guidelines – "One way to do it"
- Automatic code generator

- Help individuals and project teams
  - learn UVM and avoid pitfalls
  - become productive with UVM (saves ~ 6 weeks)
  - use UVM consistently
- Reduce the burden of supporting UVM code

Coding Guidelines

Consistency
Avoiding pitfalls
Reusability
High quality verification

Naming conventions and code structure	Where to do certain things
Restrictions on feature usage	Macros, policy objects, pre/post_body
Canonical structure for the component hierarchy	Envs, agents, subscribers, ports/exports, interfaces
Consistent pattern for writing classes and instantiating objects	Components, transaction, sequences, factory
Specific way to use configuration objects	Top-down build
Specific way to use sequences and tests	For constrained random

www.doulos.com

Home > Knowhow > Sysverilog > Uvm > Easier Uvm Guidelines > Easier UVM - Summary

Summary of the Easier UVM Coding Guidelines

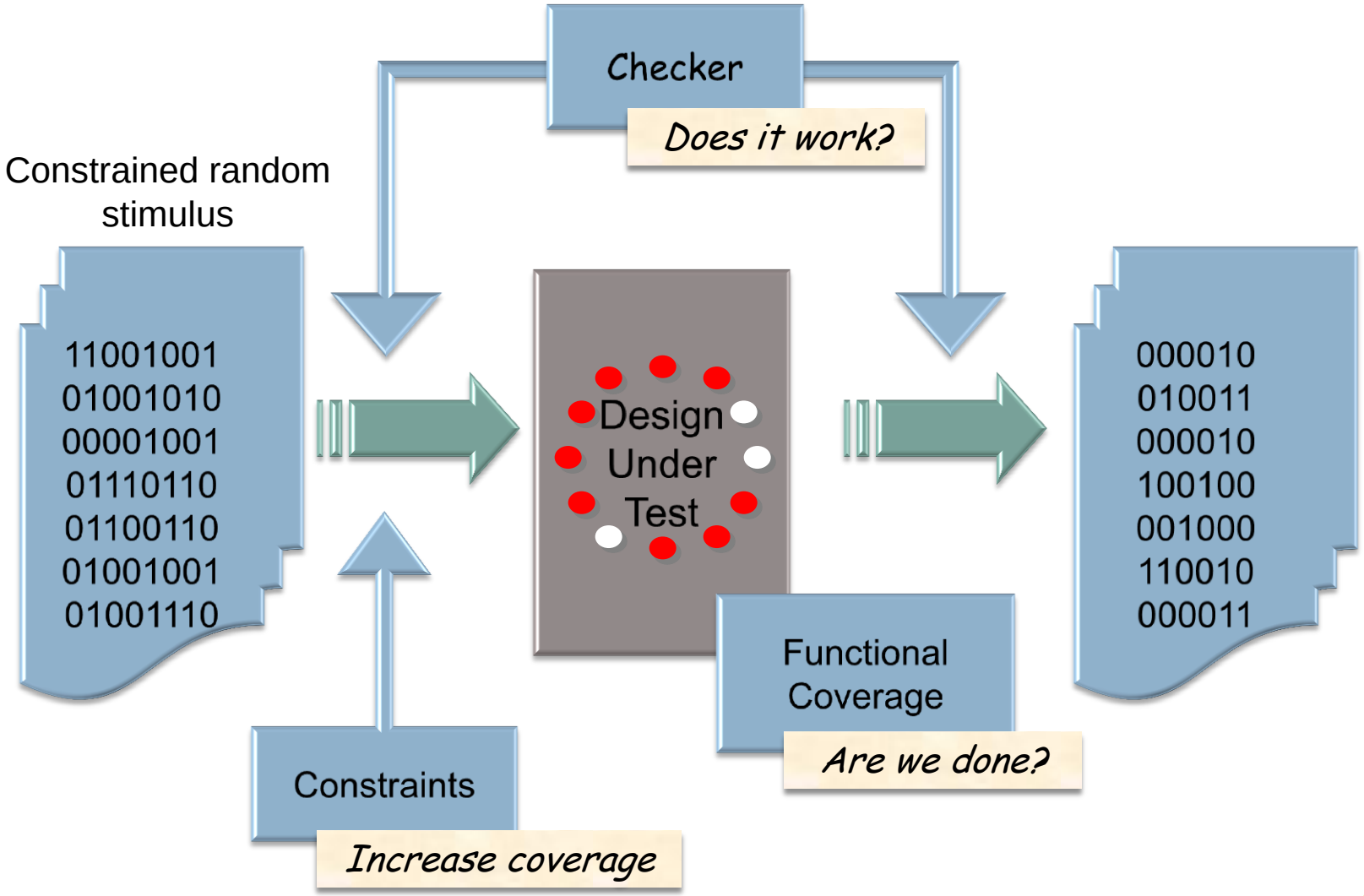
Version 2014-09-08

Lexical Guidelines and Naming Conventions

☐ Have only one declaration or statement per line.

☐ When creating user-defined names for SystemVerilog variables and classes, use lower-case words separated by underscores (as opposed to camelBackStyle).

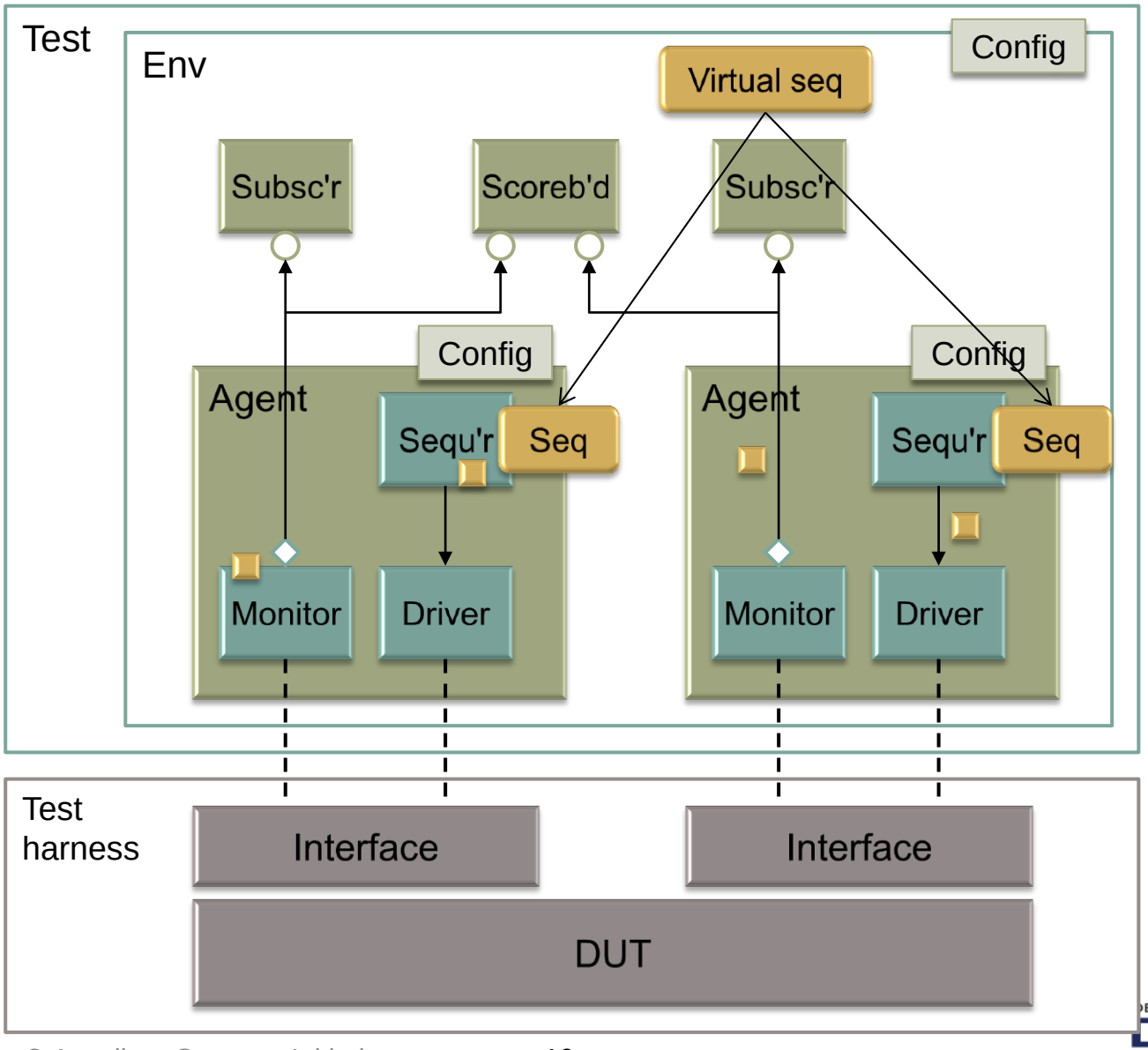
Constrained Random Verification



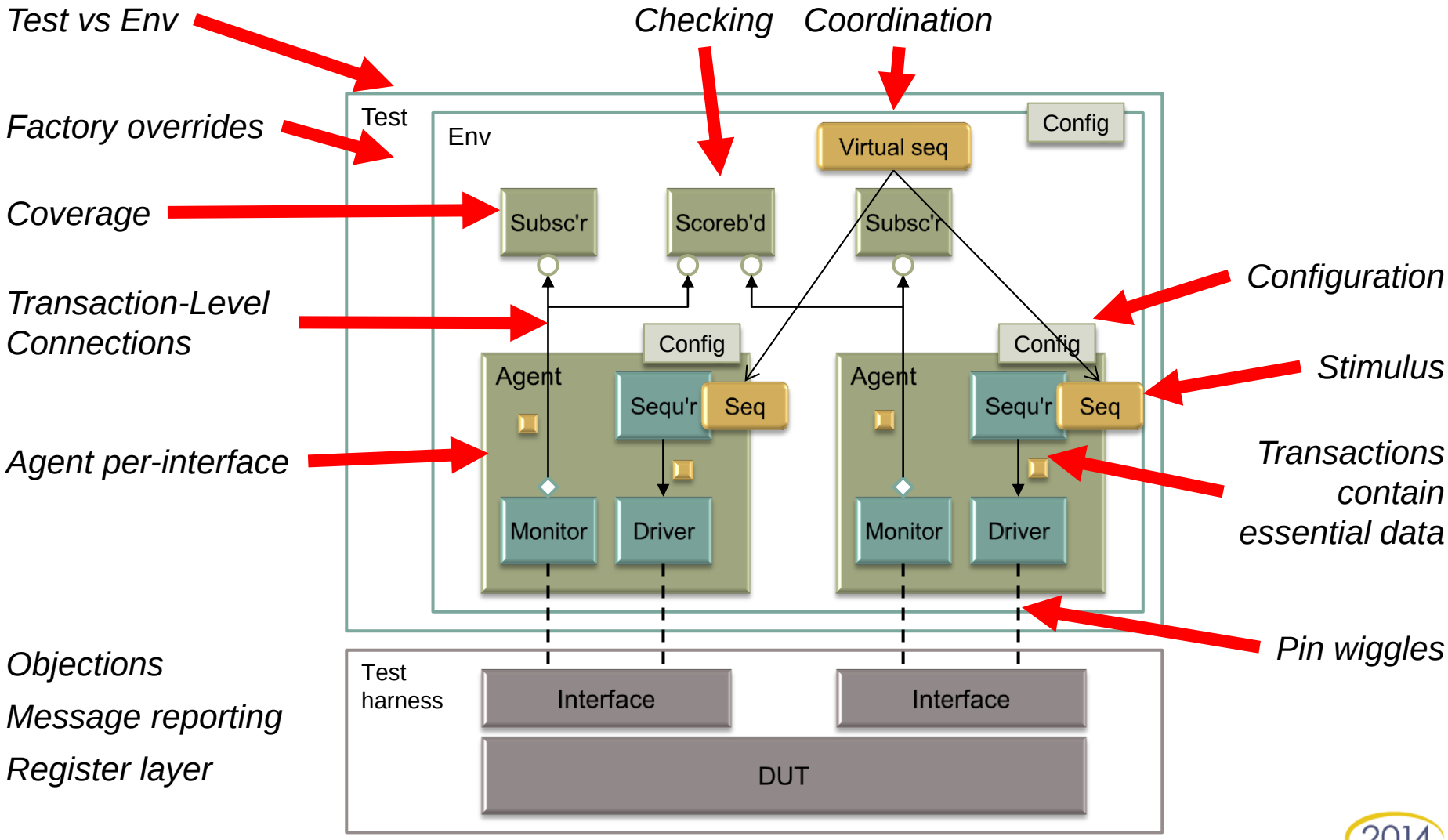
UVM Verification Environment

Class-based

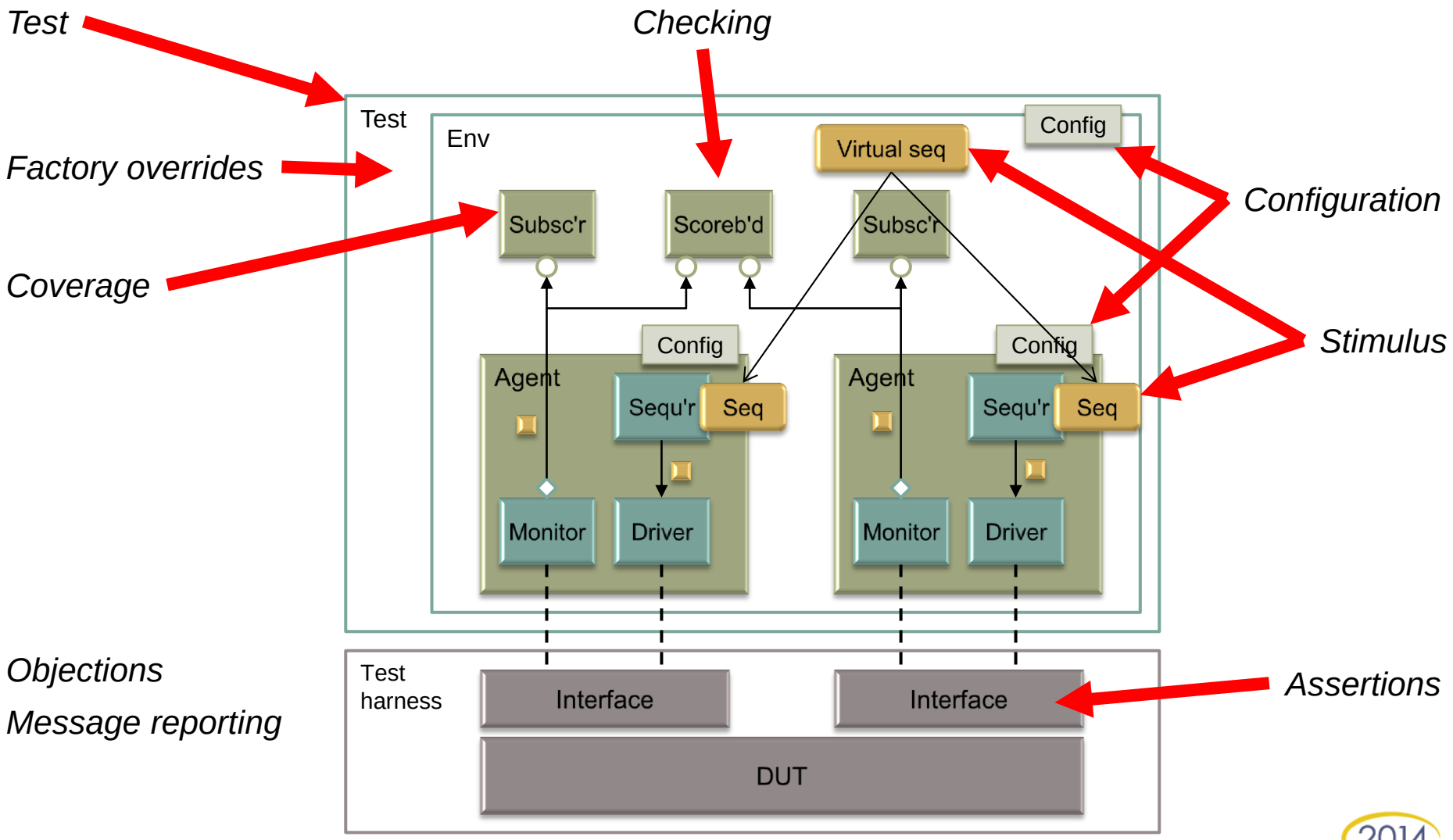
Module-based



UVM Highlights



As a Test Writer ...



Quasi-static vs Dynamic Objects

Quasi-static vs Dynamic Objects

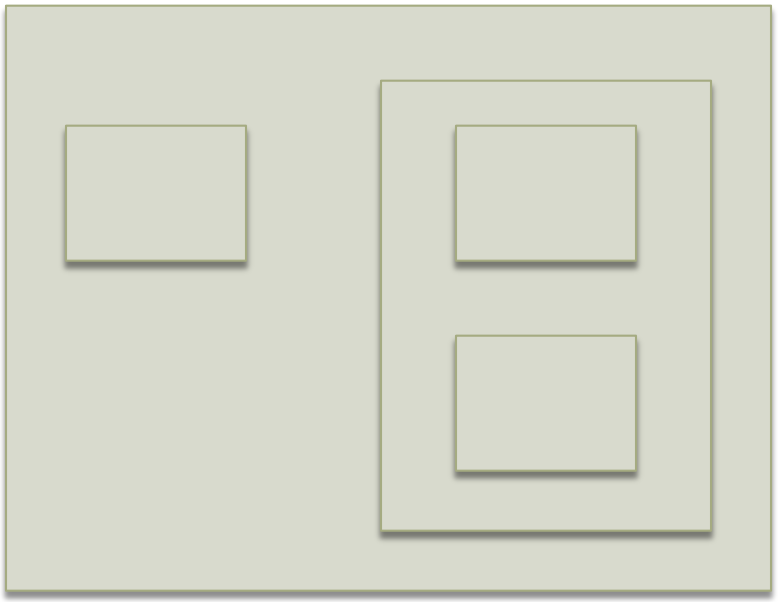
Structure

Component

Quasi-static

Stimulus, data

Transaction



Quasi-static vs Dynamic Objects

Structure

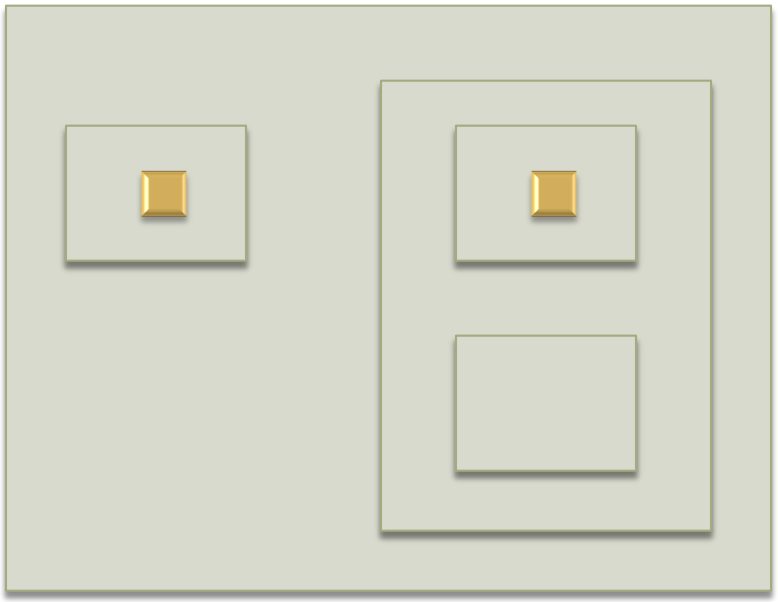
Component

Quasi-static

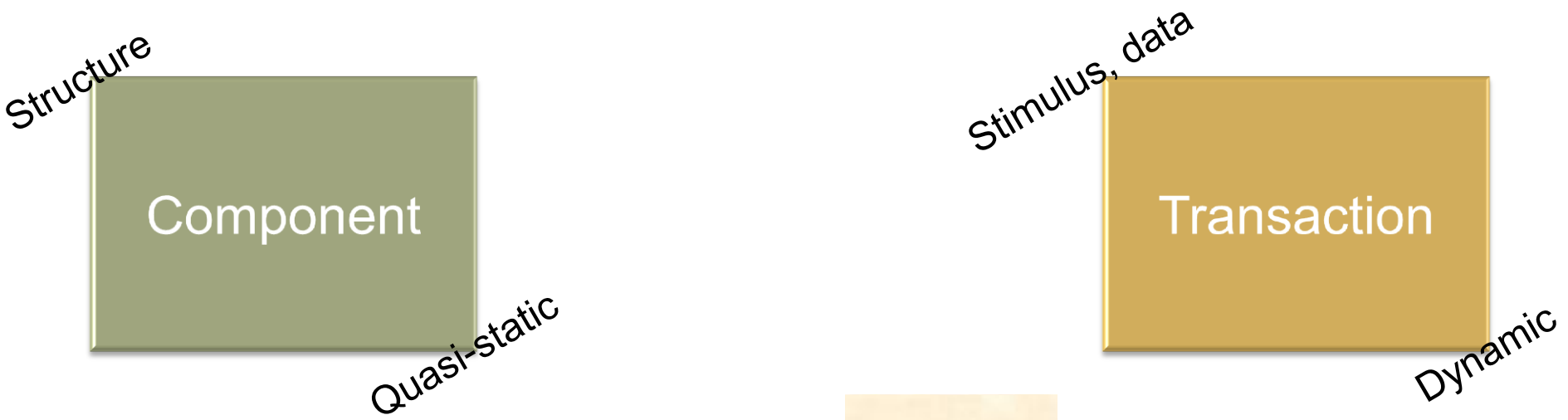
Stimulus, data

Transaction

Dynamic



Quasi-static vs Dynamic Objects



Pattern 1

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)

  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction
  ...
endclass
```

Pattern 2

```
class my_tx extends uvm_sequence_item;
  `uvm_object_utils(my_tx)

  function new (string name = "");
    super.new(name);
  endfunction
  ...
endclass
```

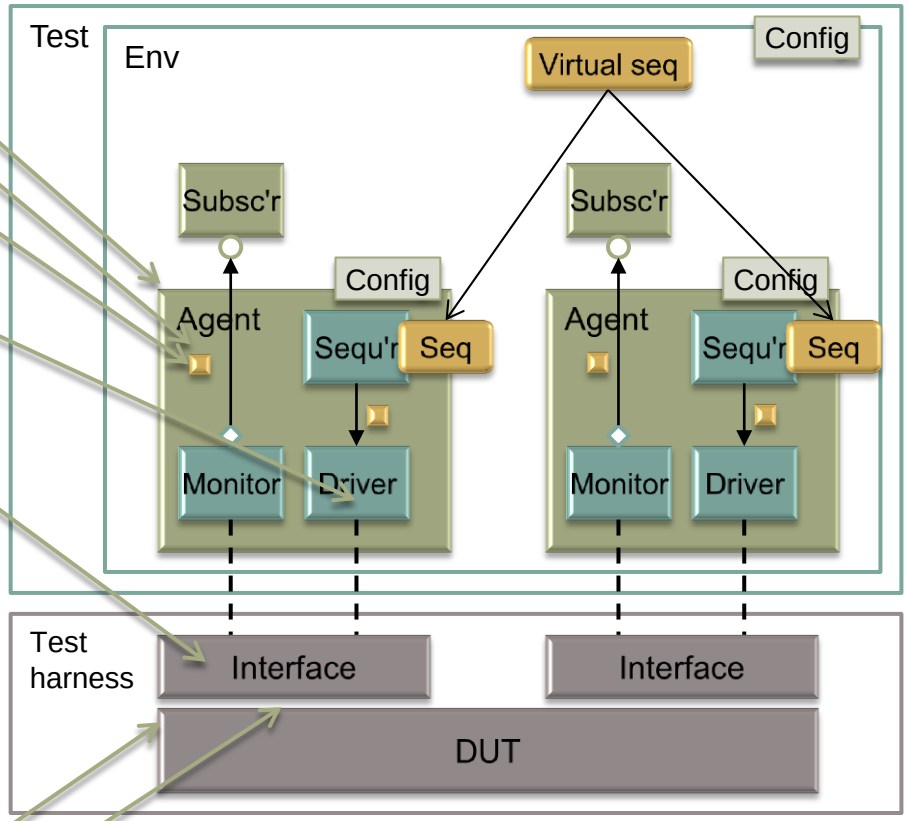
Easier UVM Code Generator Files

`clkndata.tpl`	Interface template file
`common.tpl`	Common template file
`pinlist`	Pin list file
`mydut/files.f`	List of SystemVerilog files and command line flags
`mydut/mydut.sv`	SystemVerilog source file for the DUT
`include/clkndata_do_drive.sv`	Implementation of driver
`gen`	Script to run the code generator
`run`	Script to run the simulator

Template Files

Interface template file

```
agent_name = clkndata
trans_item = data_tx
trans_var  = rand byte data;

driver_inc = clkndata_do_drive.sv

if_port  = logic clk;
if_port  = byte data;
if_clock = clk
```



Common template file

```
dut_source_path = mydut
inc_path = include
dut_top = mydut
dut_pfile = pinlist
```

Transaction Class

```
`ifndef CLKNDATA_SEQ_ITEM_SV
`define CLKNDATA_SEQ_ITEM_SV

class data_tx extends uvm_sequence_item;
  `uvm_object_utils(data_tx)

  rand byte data;

  extern function new(string name="");
  extern function void do_copy(uvm_object rhs);
  extern function bit  do_compare(uvm_object rhs, uvm_comparer comparer);
  extern function string convert2string();
  extern function void do_print(uvm_printer printer);
  extern function void do_record(uvm_recorder recorder);

endclass : data_tx
```

Interface template file

```
trans_item = data_tx
trans_var  = rand byte data;
```

Transaction Methods

```
function data_tx::new(string name = "");
  super.new(name);
endfunction : new

function void data_tx::do_copy(uvm_object rhs);
  data_tx rhs_;
  if(!$cast(rhs_, rhs))
    `uvm_fatal("do_copy", "cast of rhs object failed")
  super.do_copy(rhs);
  data = rhs_.data;
endfunction : do_copy

function bit data_tx::do_compare(uvm_object rhs, uvm_comparer comparer);
  data_tx rhs_;
  if(!$cast(rhs_, rhs))
    `uvm_fatal("do_copy", "cast of rhs object failed")
  return super.do_compare(rhs, comparer) &&  data == rhs_.data;
endfunction : do_compare

function string data_tx::convert2string();
  string s;  $sformat(s, "%s\n", super.convert2string());
  $sformat(s, "%s\n  data = \t%0h\n", get_full_name(),  data);
  return s;
endfunction : convert2string
```

Standard reporting

Transaction Methods

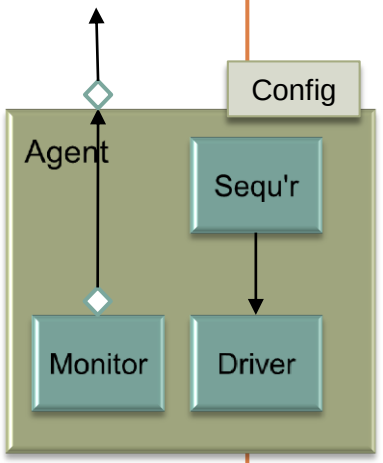
```
function void data_tx::do_print(uvm_printer printer);
  if(printer.knobs.sprint == 0)
    `uvm_info(get_type_name(), convert2string(), UVM_MEDIUM)
  else
    printer.m_string = convert2string();
endfunction : do_print

function void data_tx::do_record(uvm_recorder recorder);
  super.do_record(recorder);
  `uvm_record_field("data", data)
endfunction : do_record

`endif // CLKNDATA_SEQ_ITEM_SV
```

Standard reporting

Agent Class

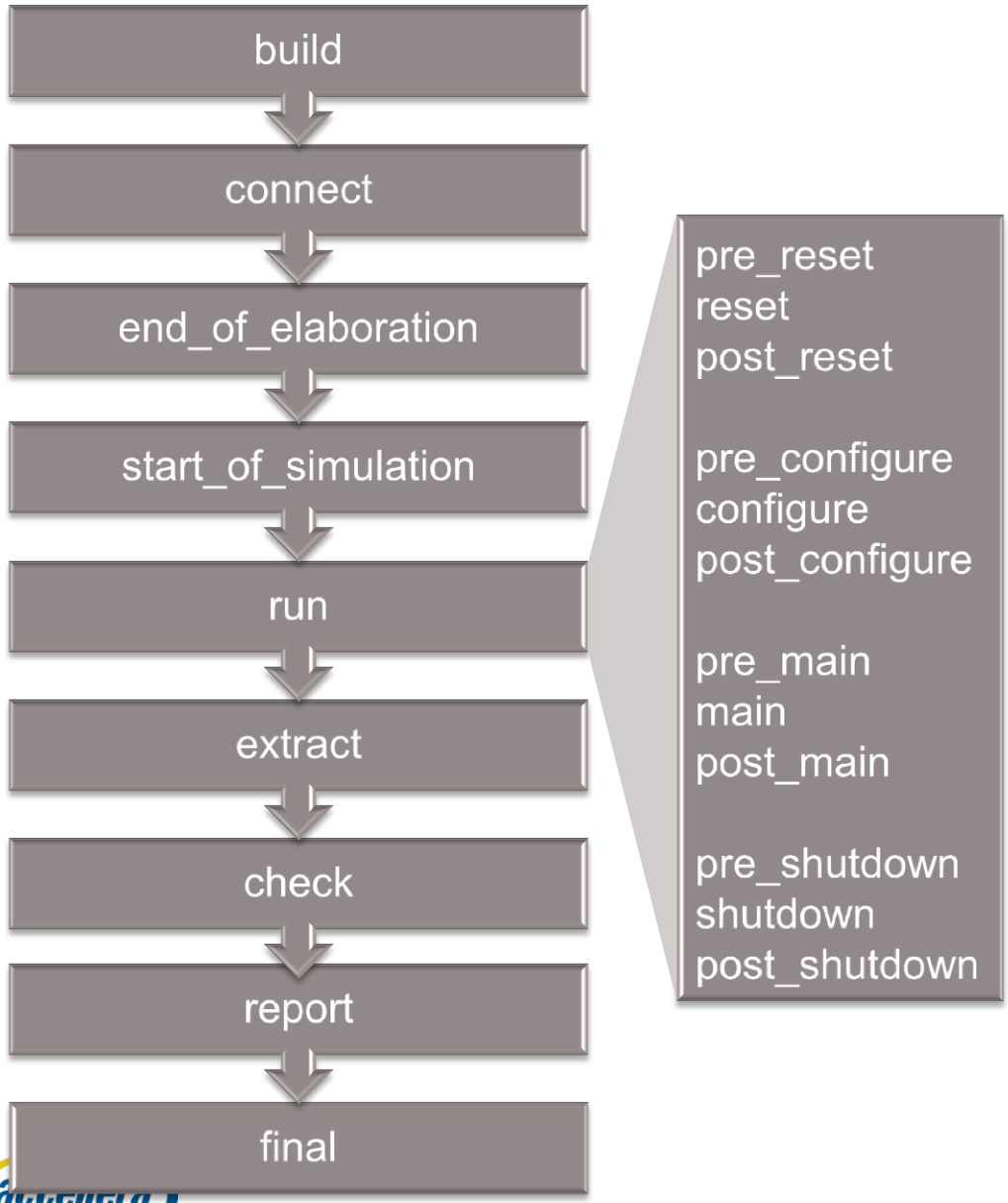
```
class clkndata_agent extends uvm_agent;
  `uvm_component_utils(clkndata_agent)

  uvm_analysis_port #(data_tx) ap;

  clkndata_config    m_cfg;
  clkndata_sequencer m_sequencer;
  clkndata_driver    m_driver;
  clkndata_monitor   m_monitor;

  extern function new(string name, uvm_component parent);
  extern function void build_phase(uvm_phase phase);
  extern function void connect_phase(uvm_phase phase);

endclass : clkndata_agent
```

TLM port

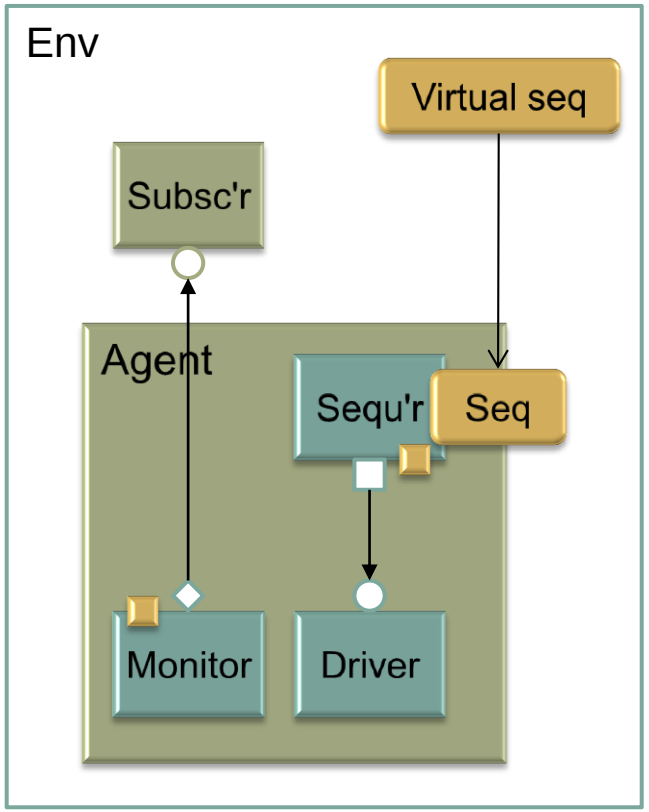



```
function clkndata_agent::new(string name, uvm_component parent);
  super.new(name, parent);
endfunction : new
```

Execution Phases



Consistent execution phases



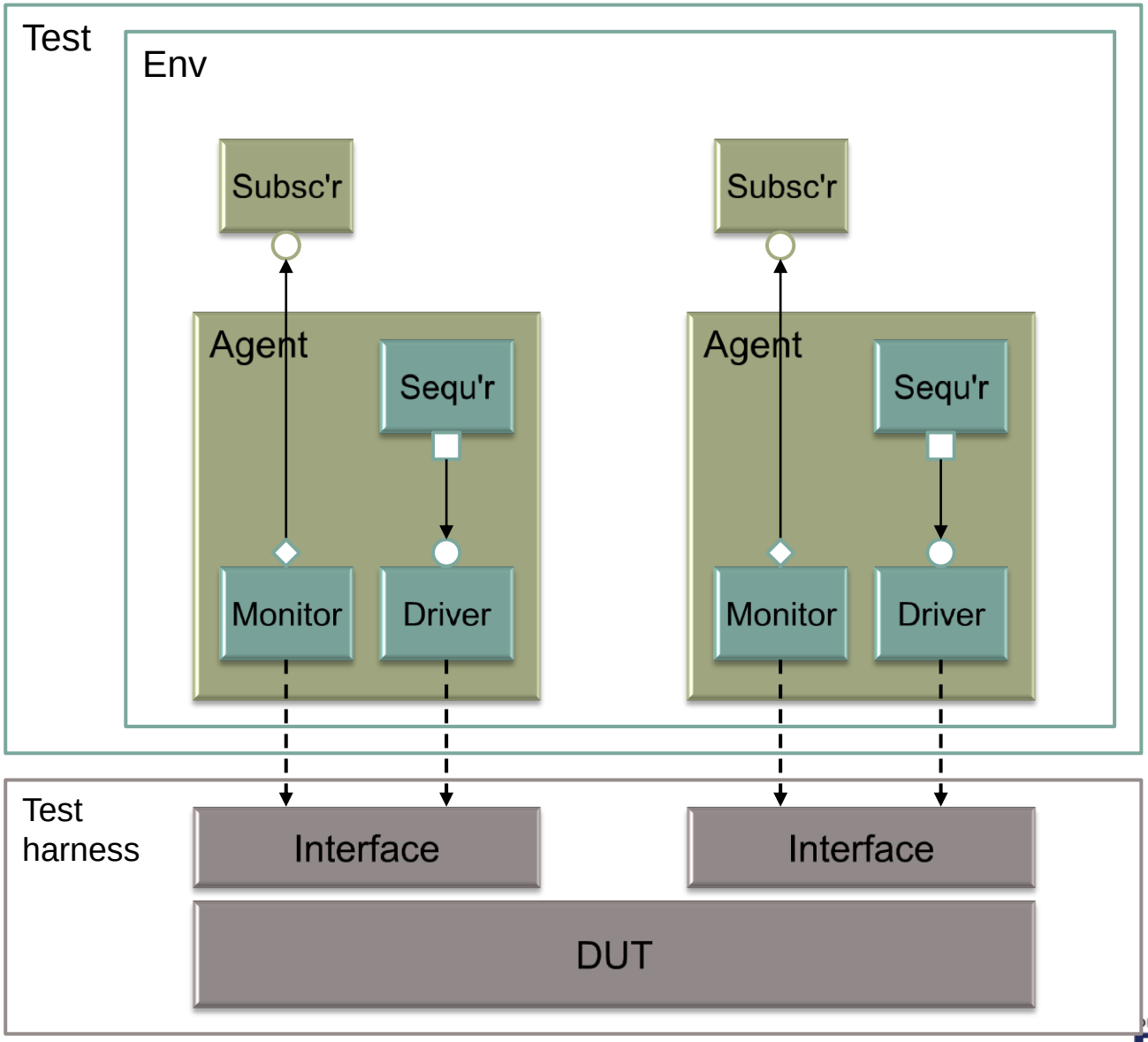
Agent Methods

```
function void clkndata_agent::build_phase(uvm_phase phase);
  if(!uvm_config_db #(clkndata_config)::get(this, "", "clkndata_config", m_cfg))
    `uvm_error(get_type_name(), "clkndata config not found")
  ...
  m_monitor = clkndata_monitor::type_id::create("m_monitor",this);
  ap = new("ap", this);
  if (m_cfg.is_active == UVM_ACTIVE)
    begin
      m_driver    = clkndata_driver   ::type_id::create("m_driver",this);
      m_sequencer = clkndata_sequencer::type_id::create("m_sequencer",this);
    end
endfunction : build_phase

function void clkndata_agent::connect_phase(uvm_phase phase);
  if(m_cfg.vif == null)
    `uvm_fatal(get_type_name(), "clkndata virtual interface is not set!")
  m_monitor.vif = m_cfg.vif;
  m_monitor.ap.connect(ap);
  if (m_cfg.is_active == UVM_ACTIVE)
    begin
      m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
      m_driver.vif = m_cfg.vif;
    end
endfunction : connect_phase
```

Active vs passive

TLM connections

Interface and Test Harness



Class-based

Virtual interfaces

Module-based

Interface and Test Harness

```
module top_th;
  timeunit 1ns;
  timeprecision 1ps;

  logic clock = 0;
  logic reset;

  clkndata_if    clkndata_if0();

  assign clkndata_if0.clk = clock;

  mydut uut (
    .clk(clkndata_if0.clk),
    .data(clkndata_if0.data)
  );

  //example clock generator process
  always #10 clock = ~clock;

  //example reset generator process
  initial
    begin
      reset=0;          //active low reset for this example
      #75 reset=1;
    end
endmodule
```

Interface template file

```
if_port  = logic clk;
if_port  = byte data;
if_clock = clk
```

```
interface clkndata_if();
  import clkndata_pkg::*;
  logic clk;
  byte data;
endinterface : clkndata_if
```

Pin list file

```
!clkndata_if
clk  clk
data data
```

Driver Class

```
class clkndata_driver extends uvm_driver #(data_tx);
  `uvm_component_utils(clkndata_driver)

  virtual clkndata_if vif;

  extern function new(string name, uvm_component parent);
  extern task run_phase(uvm_phase phase);
  extern function void report_phase(uvm_phase phase);
  extern task do_drive();

endclass : clkndata_driver
```

```
function clkndata_driver::new(string name, uvm_component parent);
  super.new(name, parent);
endfunction : new
```

Driver Methods

```
task clkndata_driver::run_phase(uvm_phase phase);
  `uvm_info(get_type_name(), "run_phase", UVM_HIGH)
  forever
    begin
      seq_item_port.get_next_item(req);
      `uvm_info(get_type_name(), {"req item\n", req.sprint}, UVM_MEDIUM)

      do_drive();

      $cast(rsp, req.clone());
      seq_item_port.item_done();
    end
endtask : run_phase
```

```
`include "clkndata_do_drive.sv"
```

```
task clkndata_driver::do_drive();
  vif.data <= req.data;
  @(posedge vif.clk);
endtask
```

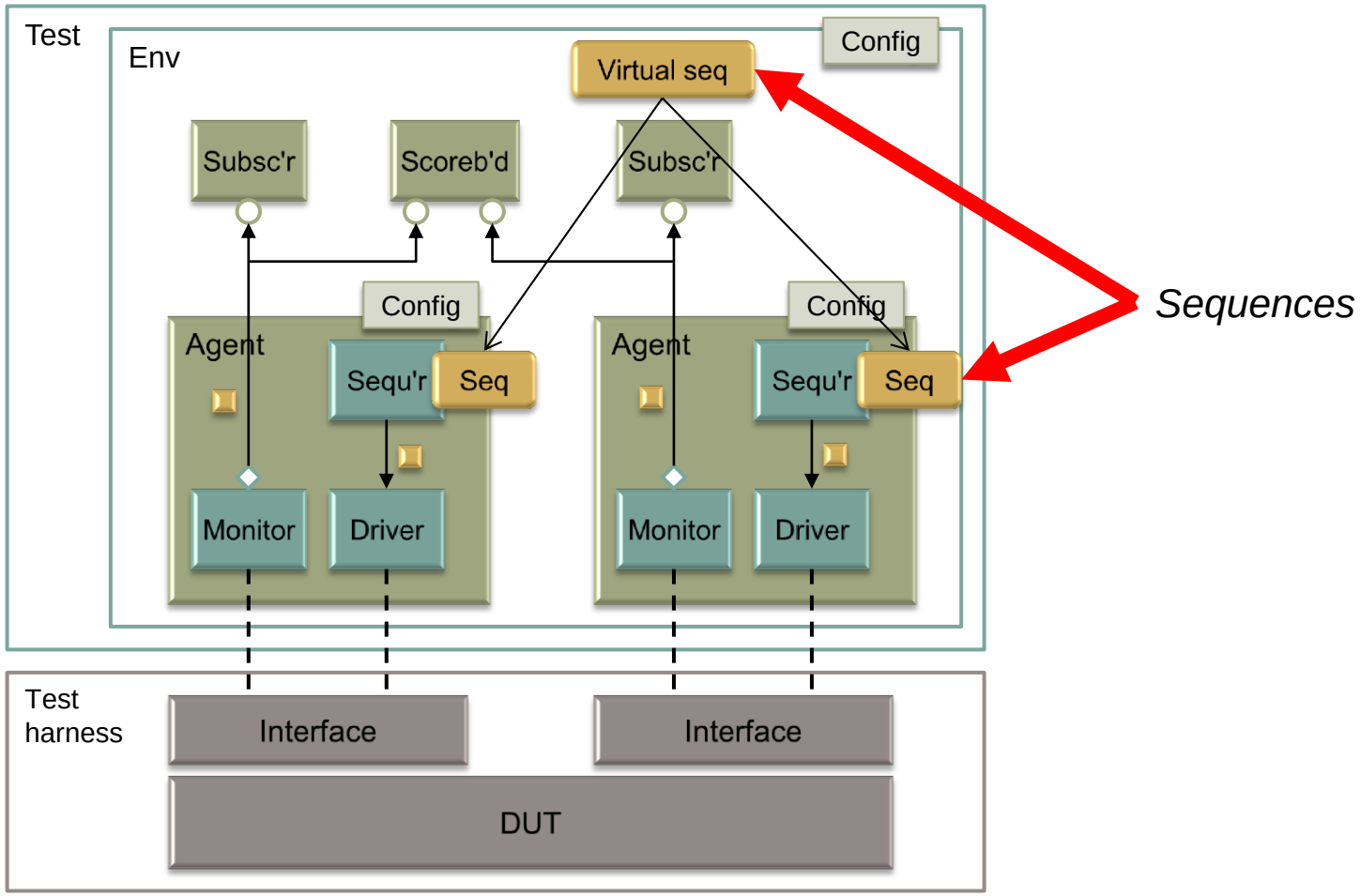
Pin wiggles

Top-Level Module

```
module top_tb;

  timeunit 1ns;
  timeprecision 1ps;

  import uvm_pkg::*;
  `include "uvm_macros.svh"

  import top_test_pkg::*;

  top_th th();

  initial
    begin
      uvm_config_db #(virtual clkndata_if)::set(
        null, "uvm_test_top", "clkndata_if", th.clkndata_if0);

      run_test();
    end
endmodule
```

config_db

Stimulus



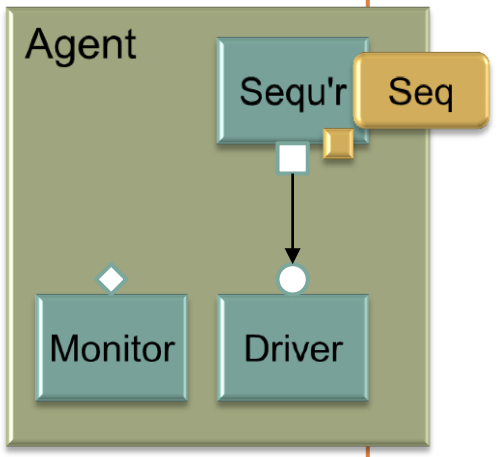
Sequence Base Class

```
class clkndata_base_seq extends uvm_sequence #(data_tx);
  `uvm_object_utils(clkndata_base_seq)

  clkndata_config  cfg;
  data_tx  item;

  extern function new(string name = "");
  extern task body();

endclass : clkndata_base_seq

function clkndata_base_seq::new(string name = "");
  super.new(name);
endfunction : new

task clkndata_base_seq::body();
  `uvm_info(get_type_name(), "clkndata_base_seq", UVM_MEDIUM)
endtask : body
```

Placeholder, going to be overridden

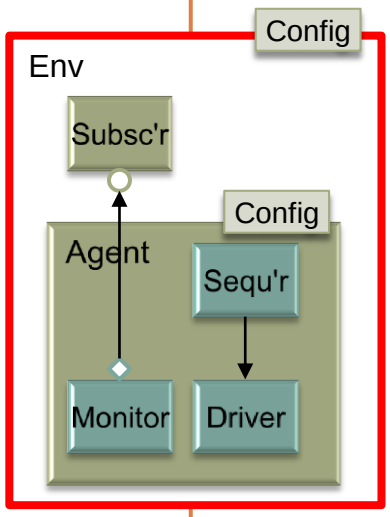
Extended Sequence Class

```
class clkndata_default_seq extends clkndata_base_seq;
  `uvm_object_utils(clkndata_default_seq)

  rand byte data;

  extern function new(string name = "");
  extern task body();

endclass : clkndata_default_seq

function clkndata_default_seq::new(string name = "");
  super.new(name);
endfunction : new

task clkndata_default_seq::body();
  `uvm_info(get_type_name(), "default sequence starting", UVM_MEDIUM)
  super.body();
  item = data_tx::type_id::create("item");
  start_item(item);
   if ( !item.randomize() )
    `uvm_warning(get_type_name(), "randomization failed!")
  finish_item(item);
  `uvm_info(get_type_name(), "default sequence completed", UVM_MEDIUM)
endtask : body
```



Top-Level Env

```
class top_env extends uvm_env;
  `uvm_component_utils(top_env)
  clkndata_agent      m_clkndata_agent;
  clkndata_config     m_clkndata_cfg;
  clkndata_coverage   m_clkndata_coverage;
  top_config          m_cfg;
  //top_scoreboard    m_scoreboard;
  extern function new(string name, uvm_component parent);
  extern function void build_phase(uvm_phase phase);
  extern function void connect_phase(uvm_phase phase);
endclass : top_env
...
function void top_env::build_phase(uvm_phase phase);
  ...
  if (!uvm_config_db #(top_config)::get(this, "", "top_config", m_cfg))
   `uvm_error(get_type_name(), "unable to get top_config")

  m_clkndata_cfg            = new("m_clkndata_cfg");
  m_clkndata_cfg.vif        = m_cfg.clkndata_vif;
  m_clkndata_cfg.is_active = m_cfg.is_active_clkndata;
  uvm_config_db #(clkndata_config)::set(this, "m_clkndata_agent", ...
  m_clkndata_agent          = clkndata_agent   ::type_id::create(...
  m_clkndata_coverage       = clkndata_coverage::type_id::create(...
endfunction : build_phase
...
```



config_db

Configuration Class

```
class clkndata_config extends uvm_object;
  // Don't register with the factory!

  virtual clkndata_if vif;

  uvm_active_passive_enum is_active = UVM_ACTIVE;

  extern function new(string name = "");
   //You may add more arguments to the constructor

endclass : clkndata_config

function clkndata_config::new(string name = "");
  super.new(name);
endfunction : new
```

Summary of Coding Idioms

Pattern 1

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(...);
    ...
endclass
```

Pattern 2a

```
class my_tx extends uvm_sequence_item;
  `uvm_object_utils(my_tx)

  function new (string name = "");
    super.new(name);
  endfunction

  function bit do_compare(...);
    ...
endclass
```

Pattern 2b

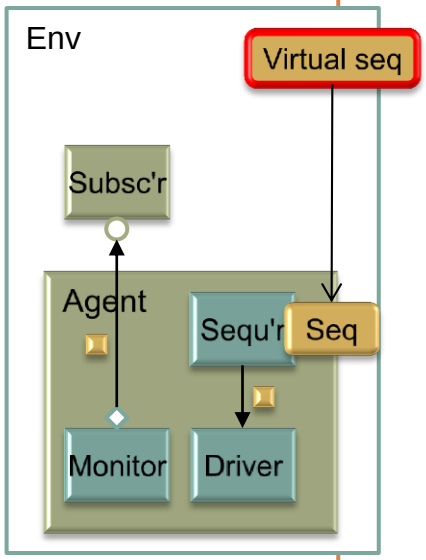
```
class my_seq extends uvm_sequence #(my_tx);
  `uvm_object_utils(my_seq)

  function new(string name = "");
    super.new(name);
  endfunction
  ...
  task body;
    ...
endclass
```

Pattern 2c

```
class my_config extends uvm_object;
  `uvm_object_utils(my_config)

  function new(string name = "");
    super.new(name);
  endfunction
  ...
endclass
```

Virtual Sequence

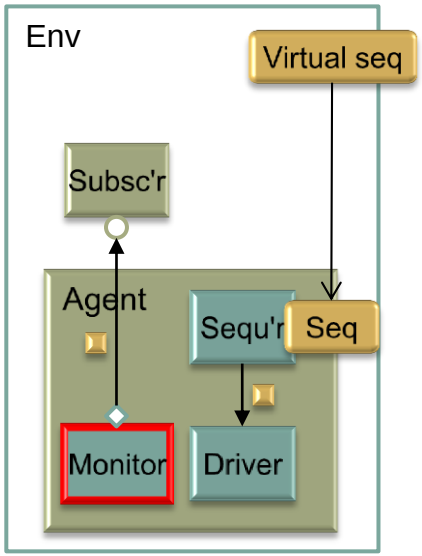
```
class top_default_seq extends top_base_seq;
  `uvm_object_utils(top_default_seq)

  extern function new(string name = "");
  extern task body();
endclass : top_default_seq
...

task top_default_seq::body();
  super.body();
  repeat(m_seq_count)
    begin
      fork
        if (m_clkndata_agent.m_cfg.is_active == UVM_ACTIVE)
          begin
            clkndata_base_seq seq;
            seq = clkndata_base_seq::type_id::create("seq");
            seq.randomize();
            seq.start(m_clkndata_agent.m_sequencer, this);
          end
        ... // Other agents
      join
      `uvm_info(get_type_name(), "default sequence completed", UVM_MEDIUM)
    end
endtask : body
```



Monitor Class

```
class clkndata_monitor extends uvm_monitor;
  `uvm_component_utils(clkndata_monitor)

  virtual clkndata_if vif;
  uvm_analysis_port #(data_tx) ap;
  clkndata_config  cfg;
  data_tx trans;

  extern function new(string name, uvm_component parent);
  extern function void build_phase(uvm_phase phase);
  extern task run_phase(uvm_phase phase);
  extern task do_mon();
endclass : clkndata_monitor
...
task clkndata_monitor::run_phase(uvm_phase phase);
  data_tx trans_clone;
  `uvm_info(get_type_name(), "run_phase", UVM_MEDIUM)
  trans = data_tx::type_id::create("trans");
  do_mon();
endtask : run_phase
```

TLM port

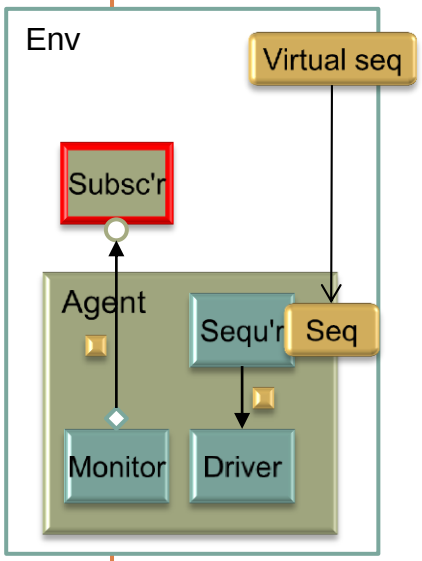



```
task clkndata_monitor::do_mon();
  forever @(posedge vif.clk)
  begin
    trans.data = vif.data;
    ap.write(trans);
  end
endtask : do_mon
```

User-defined

Subscriber Class

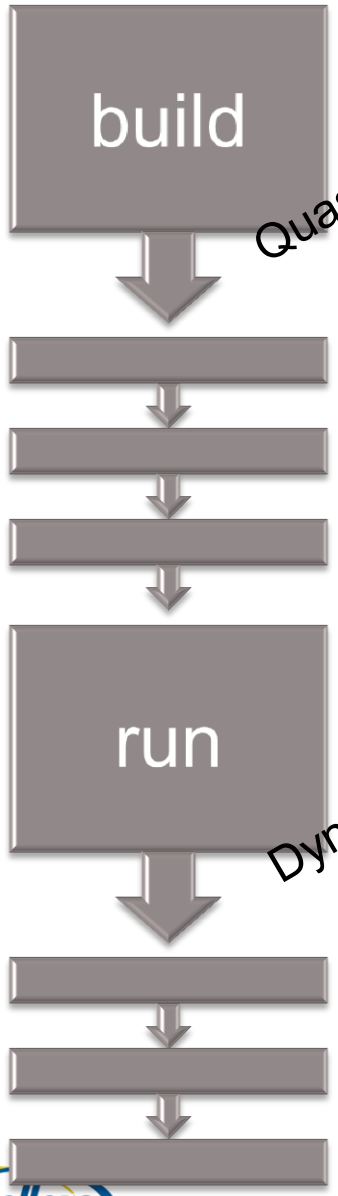
```
class clkndata_coverage extends uvm_subscriber #(data_tx);
  `uvm_component_utils(clkndata_coverage)
  bit is_covered;
  data_tx item;

  covergroup clkndata_cov;
    option.per_instance = 1;
    `include "clkndata_cover_inc.sv"
  endgroup

  extern function new(string name, uvm_component parent);
  extern function void write(input data_tx t);
  extern function void report_phase(uvm_phase phase);
endclass : clkndata_coverage
...

function void clkndata_coverage::write(input data_tx t);
  item = t;
  clkndata_cov.sample();
  if (clkndata_cov.get_inst_cover
endfunction : write
```




```
cp_data: coverpoint item.data {
  bins zero = {0};
  bins one  = {1};
  bins negative = { [-128:-1] };
  bins positive = { [1:127] };
  option.at_least = 16;
}
```

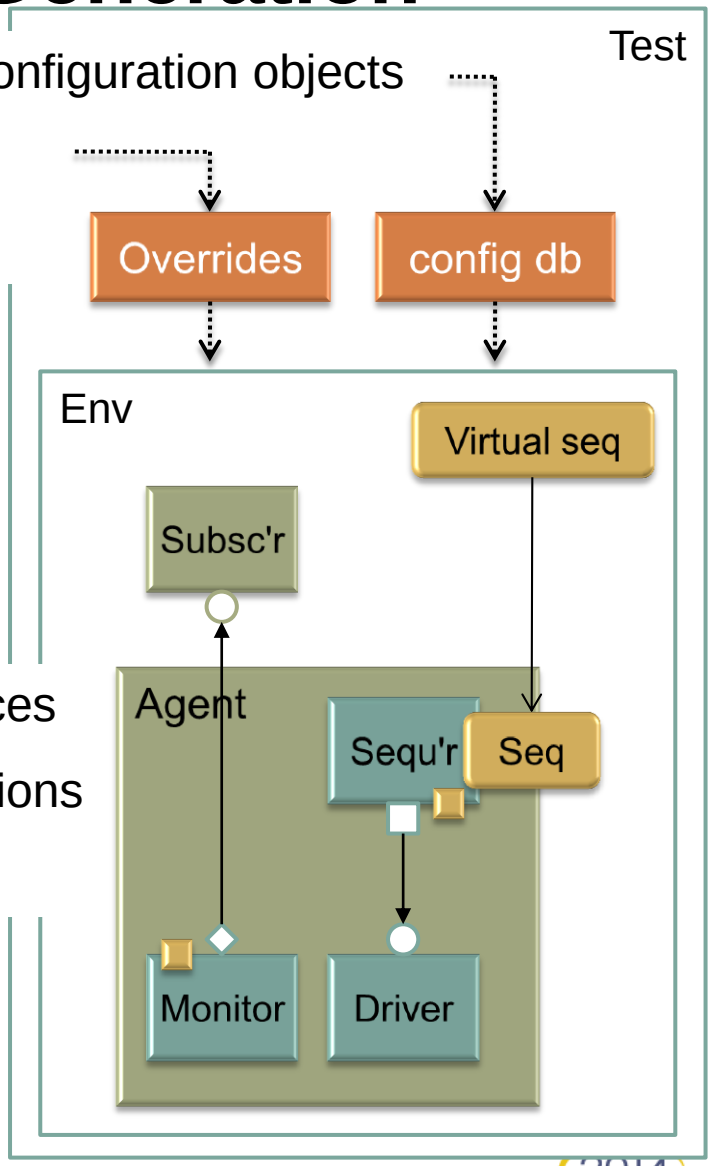
User-defined

Constrained Random Generation



- Populate/randomize configuration objects
- Set factory overrides
- Top-down build

- Factory-made sequences
- Factory-made transactions
- Constraints



Test Base Class

```
class top_base_test extends uvm_test;
  `uvm_component_utils(top_base_test)

  top_config m_cfg;
  top_env    m_env;

  extern function new(string name, uvm_component parent);
  extern function void build_phase(uvm_phase phase);
  extern function void start_of_simulation_phase(uvm_phase phase);
endclass : top_base_test
...

function void top_base_test::build_phase(uvm_phase phase);
  m_cfg = new("m_cfg");
  if(!uvm_config_db #(virtual clkndata_if)::get(..., m_cfg.clkndata_vif) )
    `uvm_error(get_type_name(), "no virtual interface found")
  m_cfg.is_active_clkndata = UVM_ACTIVE;
  uvm_config_db #(top_config)::set(this, "m_env", "top_config", m_cfg);

  m_env = top_env::type_id::create("m_env", this);
endfunction : build_phase
```

config_db

Test Class

```
class top_test extends top_base_test;
  `uvm_component_utils(top_test)

  extern function new(string name, uvm_component parent);
  extern function void build_phase(uvm_phase phase);
  extern task run_phase(uvm_phase phase);
endclass : top_test

function top_test::new(string name, uvm_component parent);
  super.new(name, parent);
endfunction : new

function void top_test::build_phase(uvm_phase phase);

  top_base_seq::type_id::set_type_override(top_default_seq::get_type());

  clkndata_base_seq::type_id::set_type_override(
                                    clkndata_default_seq::get_type());
  super.build_phase(phase);
endfunction : build_phase
```

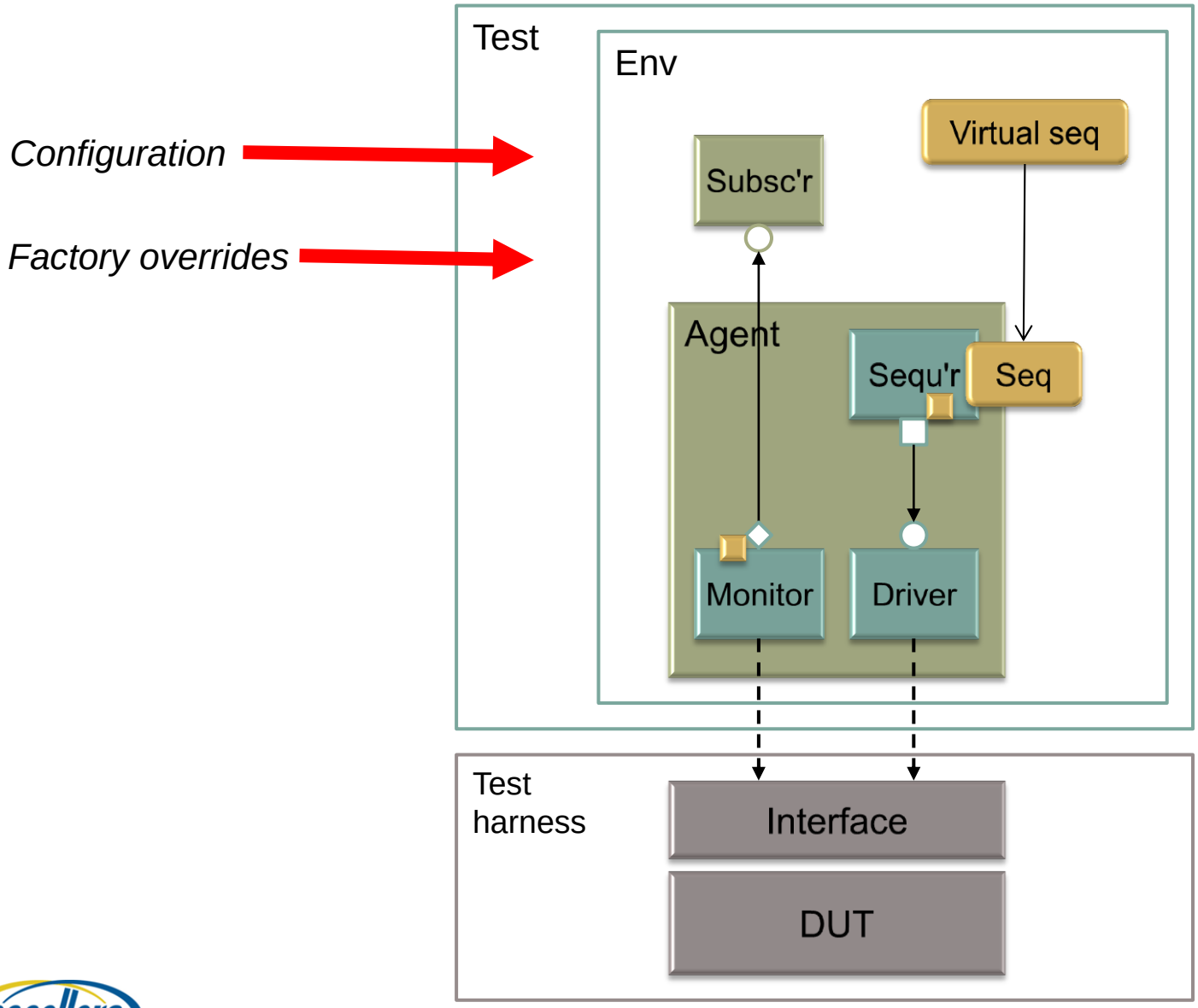
Factory overrides

Test Class

```
task top_test::run_phase(uvm_phase phase);
  top_base_seq vseq;
  vseq = top_base_seq::type_id::create("vseq");
  vseq.m_clkndata_agent = m_env.m_clkndata_agent;

  phase.raise_objection(this, "Starting virtual sequence");

  `uvm_info(get_type_name(), "Objection raised", UVM_MEDIUM)

  //uncomment and modify the following line
  //vseq.m_seq_count = 1;

  vseq.start(null);

  phase.drop_objection(this, "Finished virtual sequence");

  `uvm_info(get_type_name(), "Objection dropped", UVM_MEDIUM)
endtask : run_phase
```

Objection

Simulation Out-of-the-Box



Adding a User-Defined Sequence

```
class my_clkndata_seq extends clkndata_base_seq;
  `uvm_object_utils(my_clkndata_seq)
  rand byte data;
  extern function new(string name = "");
  extern task body();
endclass : my_clkndata_seq
...

task my_clkndata_seq::body();
  `uvm_info(...
  super.body();
  for (int i = 0; i < 16; i++)
  begin
    item = data_tx::type_id::create("item");
    start_item(item);
    if ( !item.randomize() with { data == i; })
      `uvm_warning(get_type_name(),"randomization failed!")
    finish_item(item);
     `uvm_info(...
  end
endtask : body
```

User-defined

Adding a User-Defined Sequence

Interface template file

```
agent_name = clkndata
trans_item = data_tx
trans_var  = rand byte data;

driver_inc       = clkndata_do_drive.sv
monitor_inc      = clkndata_do_mon.sv
agent_cover_inc = clkndata_cover_inc.sv
agent_seq_inc    = my_clkndata_seq.sv

agent_factory_set = clkndata_base_seq my_clkndata_seq

if_port  = logic clk;
if_port  = byte data;
if_clock = clk
```

Generated test

```
function void top_test::build_phase(uvm_phase phase);
  ...
  clkndata_base_seq::type_id::set_type_override(
                                  my_clkndata_seq::get_type());
  super.build_phase(phase);
endfunction : build_phase
```

Simulation Log

```
sed 's/UVM_VERBOSITY=UVM_FULL/UVM_VERBOSITY=UVM_NONE/' ...
```

```
...
# uvm_test_top                  top_test                  -     @471
#   m_env                       top_env                   -     @484
#     m_clkndata_agent          clkndata_agent            -     @497
#       ap                      uvm_analysis_port         -     @527
#       m_driver                clkndata_driver           -     @535
#       m_monitor               clkndata_monitor          -     @520
#         ap                    uvm_analysis_port         -     @667
#       m_sequencer             uvm_sequencer             -     @558
#     m_clkndata_coverage       clkndata_coverage         -     @504
#       analysis_imp            uvm_analysis_imp          -     @511
...
# mydut data = 00
# mydut data = 01
# mydut data = 02
# mydut data = 03
# mydut data = 04
...
```

Edited highlights

Questions?

For Further Information

http://www.doulos.com/knowhow/sysverilog/uvm