February 28 – March 3, 2011

SystemVerilog FrameWorks™ Scoreboard

An Open Source Implementation Using UVM

Dr. Ambar Sarkar

Chief Verification Technologist

Paradigm Works, Inc.

Design • Verify • Deliver ™

Agenda

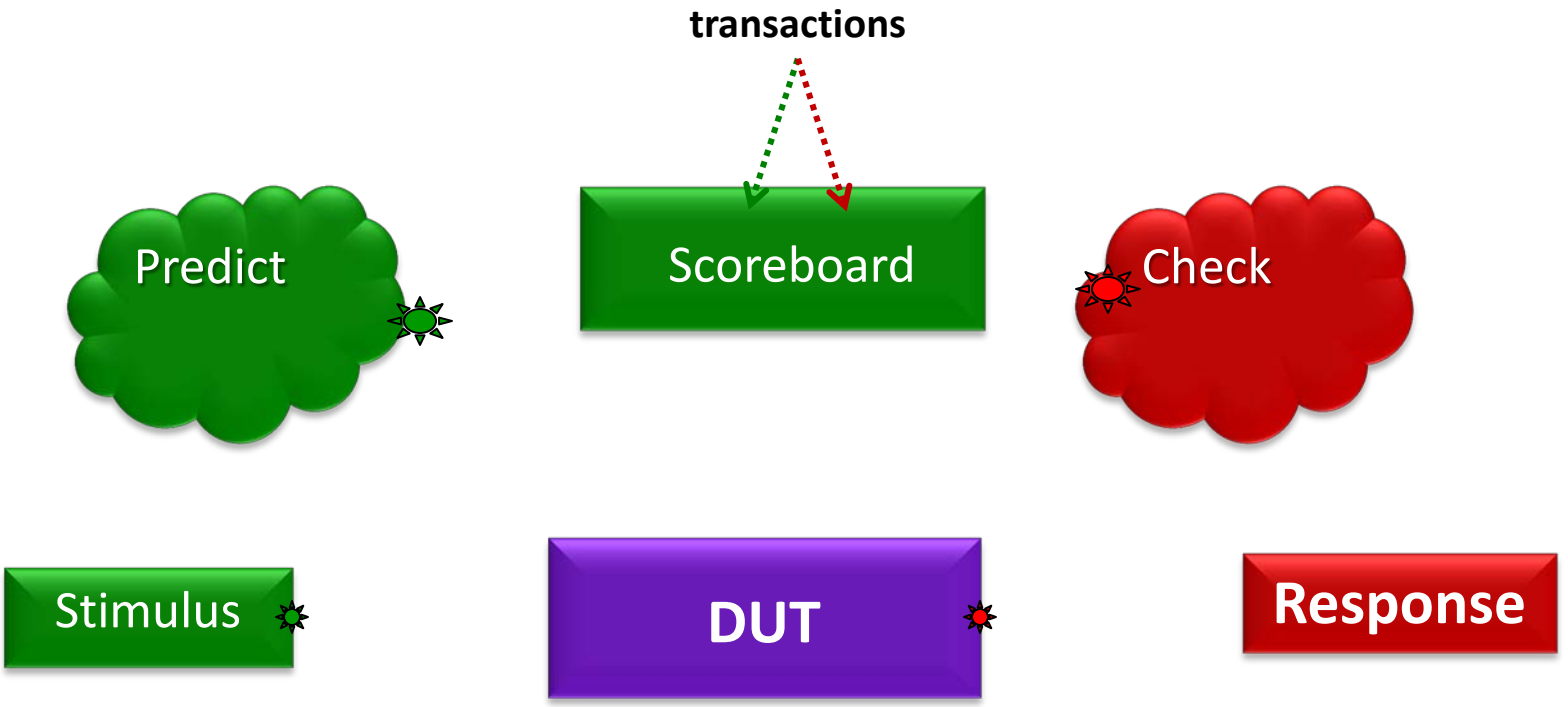
- Terminology and Requirements
- SystemVerilog Frameworks™ Scoreboard
- Use Case Examples
- Summary

Terminology and Requirements

Scoreboard Terminology

- An essential component for any self-checking constrained-random verification environment
- All implementations are variations of
  - Post - when an event of interest is anticipated by the verification environment, details of that event are posted to the scoreboard.
  - Check – When an event of interest is actually observed, it is checked against the events posted on the scoreboard.
  - Transfer function - The mapping of posted to checked events is called the transfer function
- NOT a reference model
  - But the reference model may be used to predict

What is a Scoreboard



Usage Requirements

- Work out of the box for most use cases
- Easily extendable
- Support procedural and port-based checking
- Support complex transfer functions
- Customization should scale with complexity of application
- Must be compatible with UVM methodology

emacs@GRVVTMI

File Edit Options Buffers Tools Help

```
//   permissions and limitations under the License.
//----------------------------------------------------------------------

//------------------------------------------------------------------------------
//
// CLASS: uvm_scoreboard
//
// The uvm_scoreboard virtual class should be used as the base class for
// user-defined scoreboards.
//
// Deriving from uvm_scoreboard will allow you to distinguish scoreboards from
// other component types inheriting directly from uvm_component. Such
// scoreboards will automatically inherit and benefit from features that may be
// added to uvm_scoreboard in the future.
//------------------------------------------------------------------------------

virtual class uvm_scoreboard extends uvm_component;

  // Function: new
  //
  // Creates and initializes an instance of this class using the normal
  // constructor arguments for <uvm_component>: ~name~ is the name of the
  // instance, and ~parent~ is the handle to the hierarchical parent, if any.

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

  const static string type_name = "uvm_scoreboard";

  virtual function string get_type_name ();
    return type_name;
  endfunction

endclass
```

Functional Requirements

Both in-order and out-of-order checking

Multiple streams at a time

Support for timeout checking

Dropping packets

Support segmented packets

Hooks for error handling

Reset/restart etc

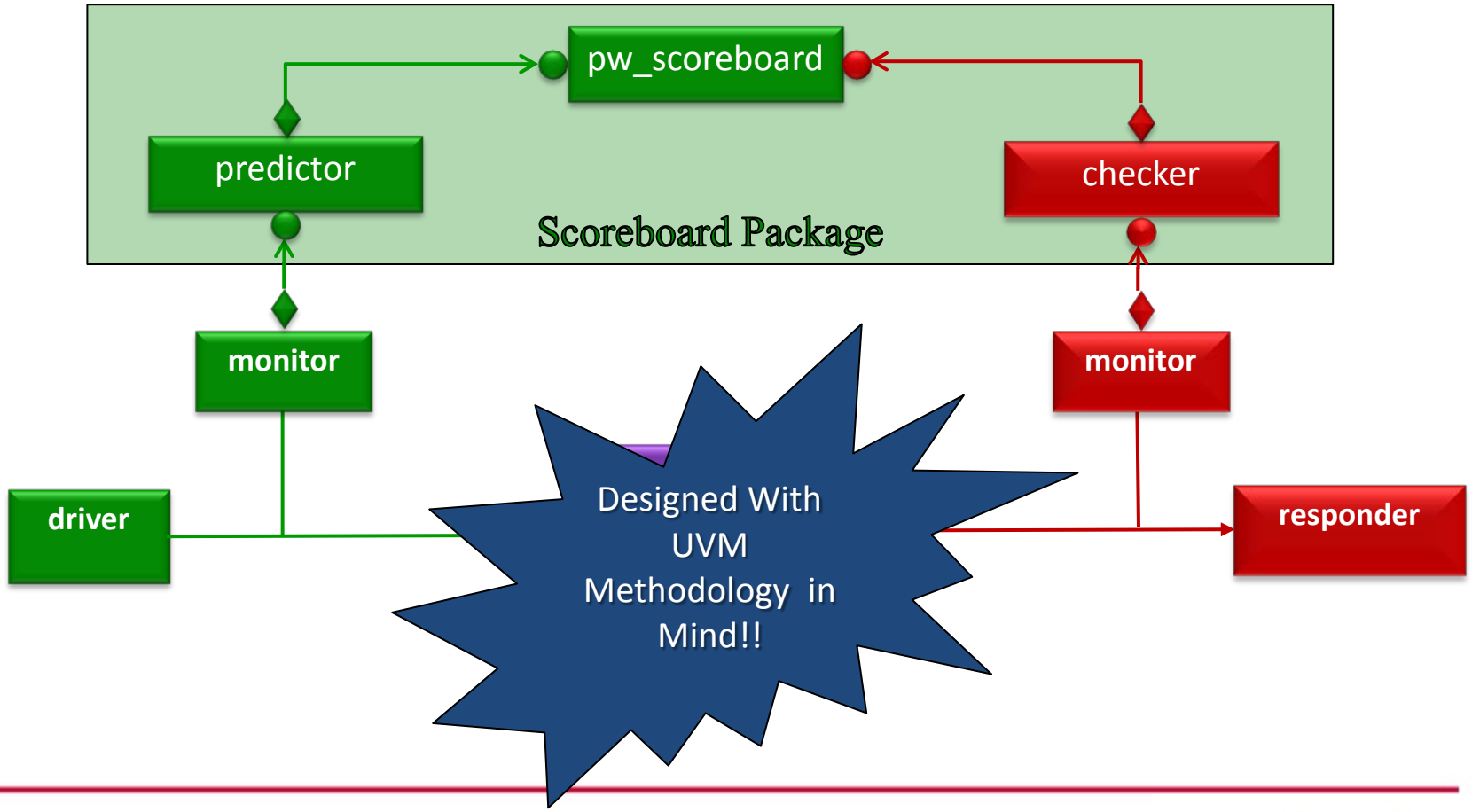
End-of-test status/statistics

Design • Verify • Deliver ™
Copyright © 2011 Paradigm Works, Inc. Proprietary & Confidential

SystemVerilog FrameWorks™ Scoreboard

SystemVerilog FrameWorks™

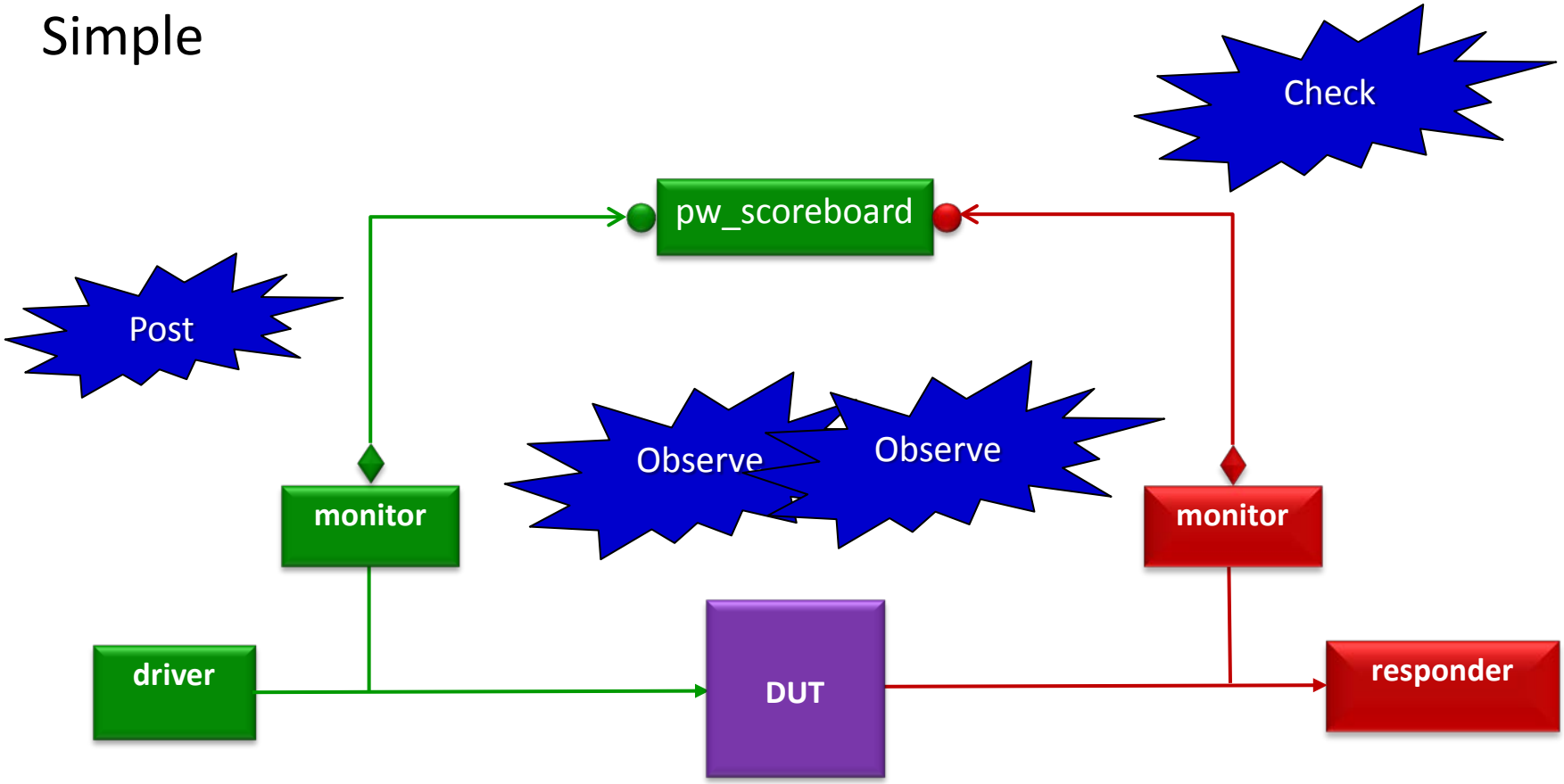
Scoreboard Package



Use Case Examples

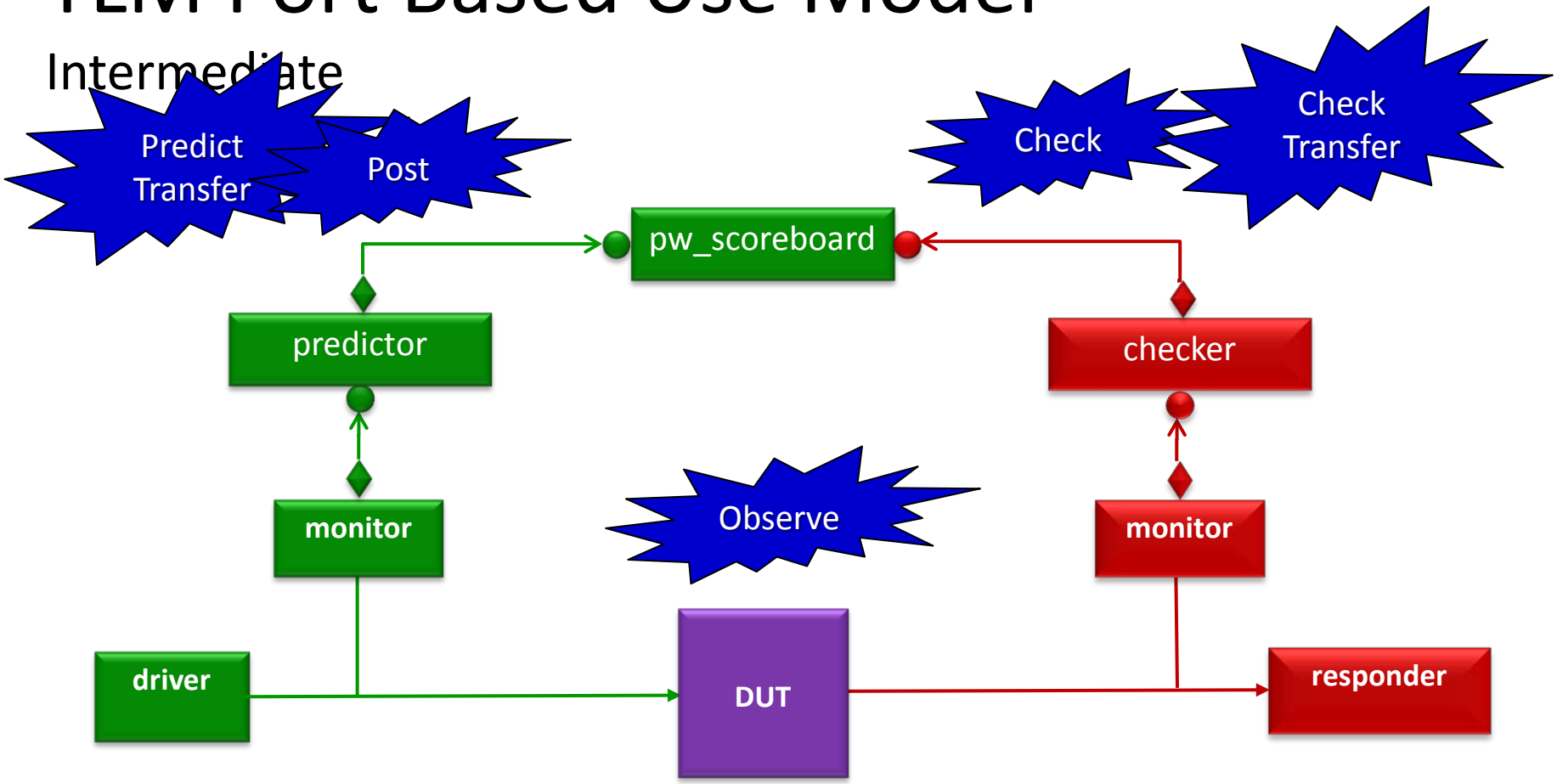
TLM Port Based Use Model

Simple



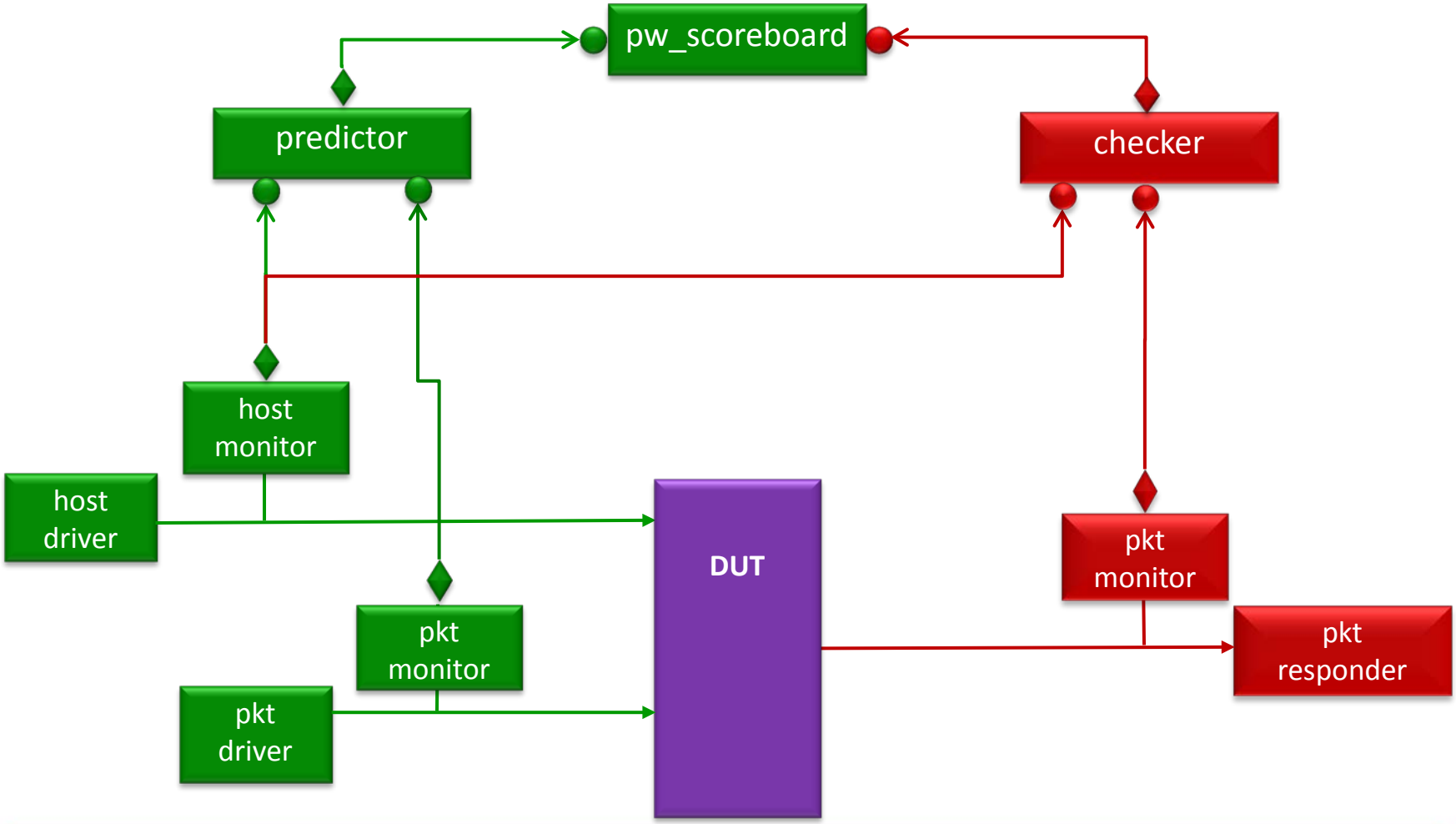
TLM Port Based Use Model

Intermediate

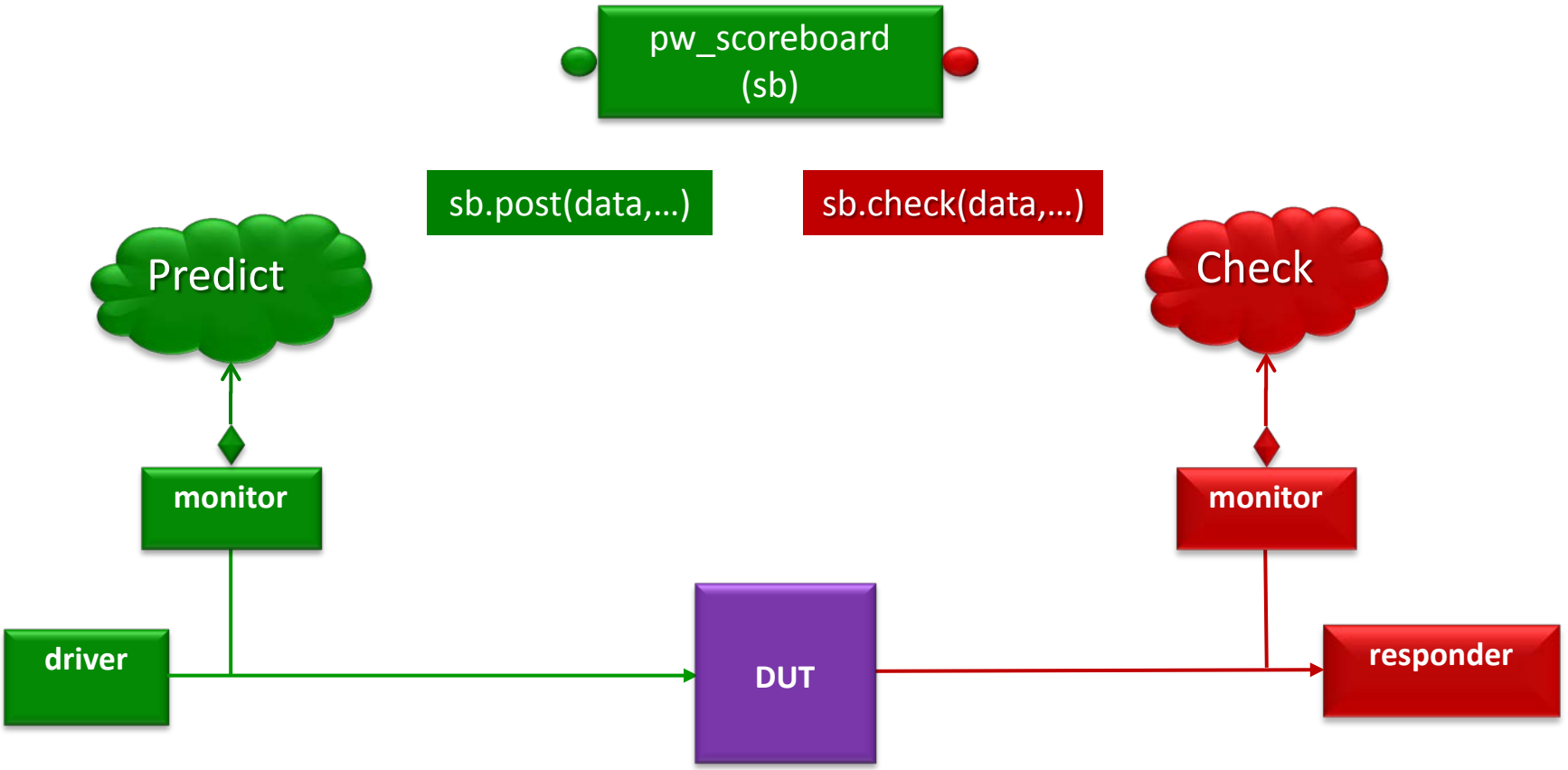


TLM Port Based Use Model

Advanced

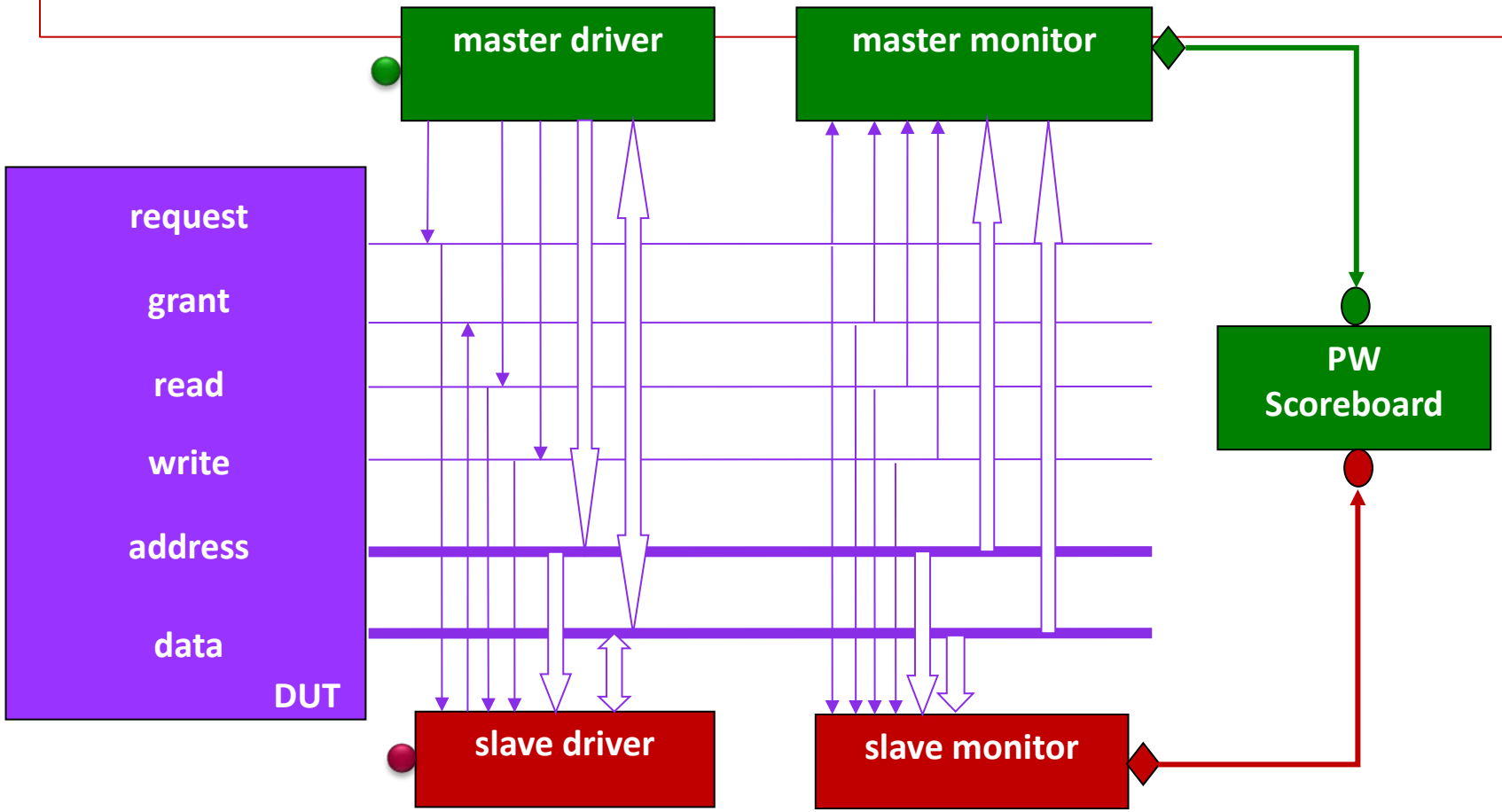


Procedural Call Based Use Model



Examples

XBUS Example



Basic Usage

```
// Include
`include "uvm.svh"
`include "pw_scoreboard_lib.svh"

// Declare typically in uvm_env
pw_scoreboard          #(xbus_transfer, xbus_transfer) sb;

// Build  typically in uvm_env::build()
sb = xbus_pw_scoreboard #(xbus_transfer, xbus_transfer)::type_id::create("sb", this);

// Connect typically in uvm_env::connect()
xbus0.masters[0].monitor.item_collected_port.connect(sb.post_export);
xbus0.slaves[0].monitor.pw_item_collected_port.connect(sb.check_export);

// Report   typically in  uvm_env::report()
sb.report_sb(  1,   // Error if outstanding
               1,  // Display outstanding
             );
```

How Is Checking Done?

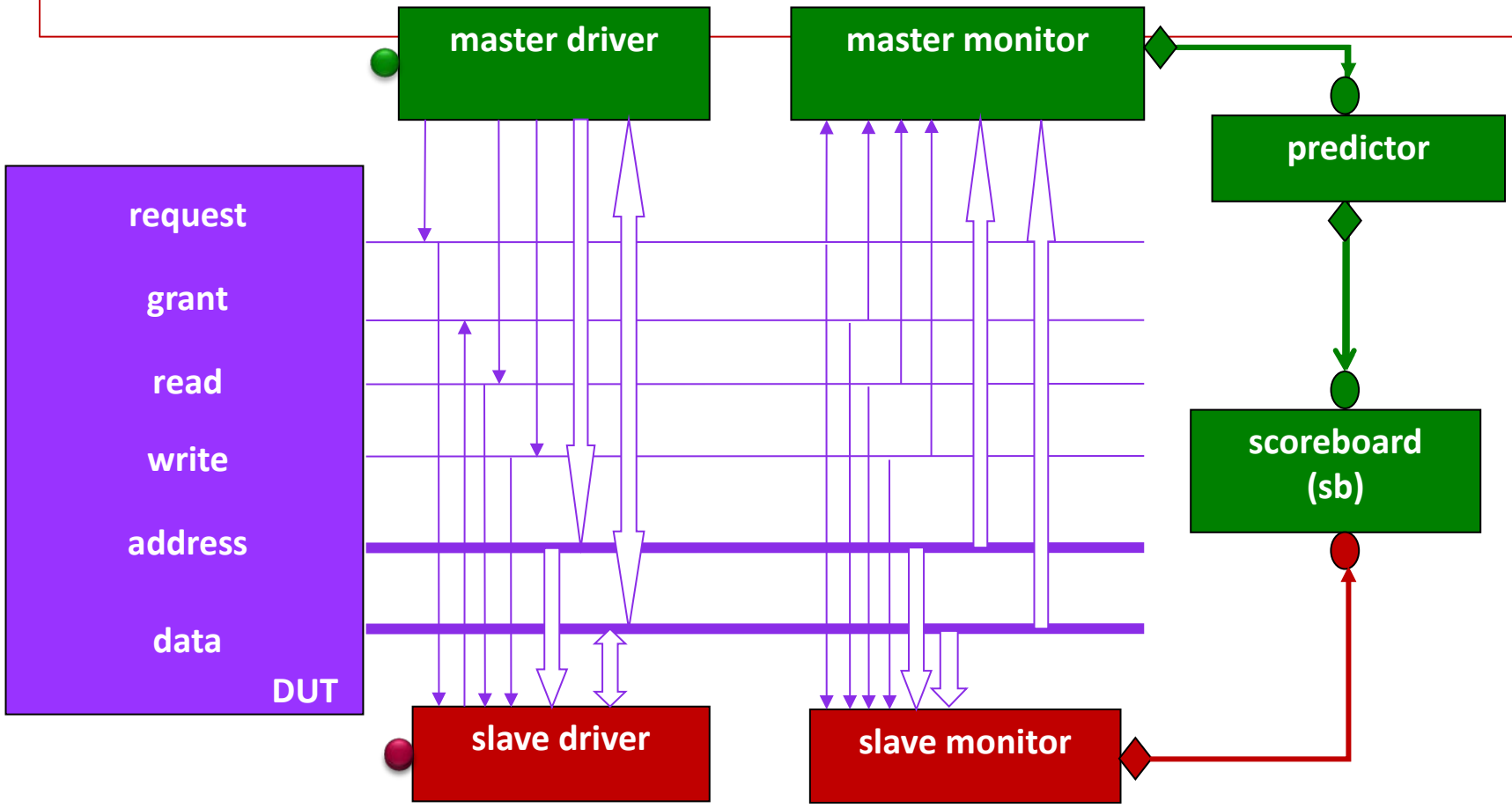
- Uses the uvm_transaction::compare()
  - Use the default compare method of the posted transaction type
  - You can override the compare policy
  - Following called internally:

```
// The actual compare taking place in the scoreboard
function bit sb_entry::comp(sb_entry rhs);
    comp = this.data.compare(rhs.data,this.comparer);
    return comp;
endfunction : comp
```

Complex Transfer Function

- Use case example:
  - Ignore packets of certain types as the dut drops them
- Steps:
  - Add predictor component
    - Derived from pw_predictor_checker
  - Override transfer task
  - Insert between monitor and scoreboard

XBUS Example



Complex Transfer Function

```
// Extend predictor base class
class xbus_predictor extends pw_predictor_checker (1,1, xbus_transfer,xbus_transfer);
  …
 // Override transfer task
  virtual task transfer(xbus_transfer trans, int port_id);
     if  (<check if trans should be posted>)
            sb_aports[0].write(trans);
  endtask : transfer
  …
endclass: xbus_predictor

// Declare typically in uvm_env
pw_scoreboard          #(xbus_transfer, xbus_transfer) sb;
xbus_predictor          predictor;
```

Complex Transfer Function

```
// Build
sb = xbus_pw_scoreboard #(xbus_transfer, xbus_transfer)::type_id::create("sb", this);
predictor = xbus_predictor::type_id::create("predict", this);

// Connect monitor to predictor
xbus0.masters[0].monitor.item_collected_port.connect(predictor.inp_exports[0]);
xbus0.slaves[0].monitor.pw_item_collected_port.connect(sb.inp_exports[0]);

// Connect predictor to scoreboard
predictor.sb_aports[0].connect(sb.post_export);
checker.sb_aports[0].connect(sb.check_export);
```

Dropping Packets

- Use case example (Error injection scenarios):
  - Allow a few packets to drop in a time window
  - Hard to predict which ones will drop
  - Do not drop too many!
- Steps:
  - Override pw_scoreboard::get_canDrop()
  - Set allowed # of droppable packets

Dropping Packets

```
// Extend
class acme_pw_scoreboard extends pw_scoreboard;
   `uvm_component_utils_begin(acme_pw_scoreboard)
      `uvm_field_int(disable_scoreboard, UVM_ALL_ON)
      `uvm_field_int(droppable_cnt, UVM_ALL_ON)
   `uvm_component_utils_end
 …
   // Specify when okay to drop. This is called when about to check
   // posted is the originally posted data
    virtual function int get_canDrop(uvm_transaction posted);
        return ($time < 200ns && $time > 300ns);
    endfunction: get_canDrop
…
   // Build  typically in uvm_env::build()
   sb = acme_pw_scoreboard #(xbus_transfer, xbus_transfer)::type_id::create("sb", this);
…
  // Allow a maximum of 3 packets to drop
   set_config_int("xbus_demo_tb0.sb",“droppable_cnt",3);
```

Multiple Streams- Approach 1

- Use case example:
  - Multiple independent streams/flows
  - In-order within stream
  - Out-of-order in between
  - Arbitrary number of streams/flows
- Steps
  - Extend pw_scoreboard::get_stream_id()

Multiple Streams – Approach 1

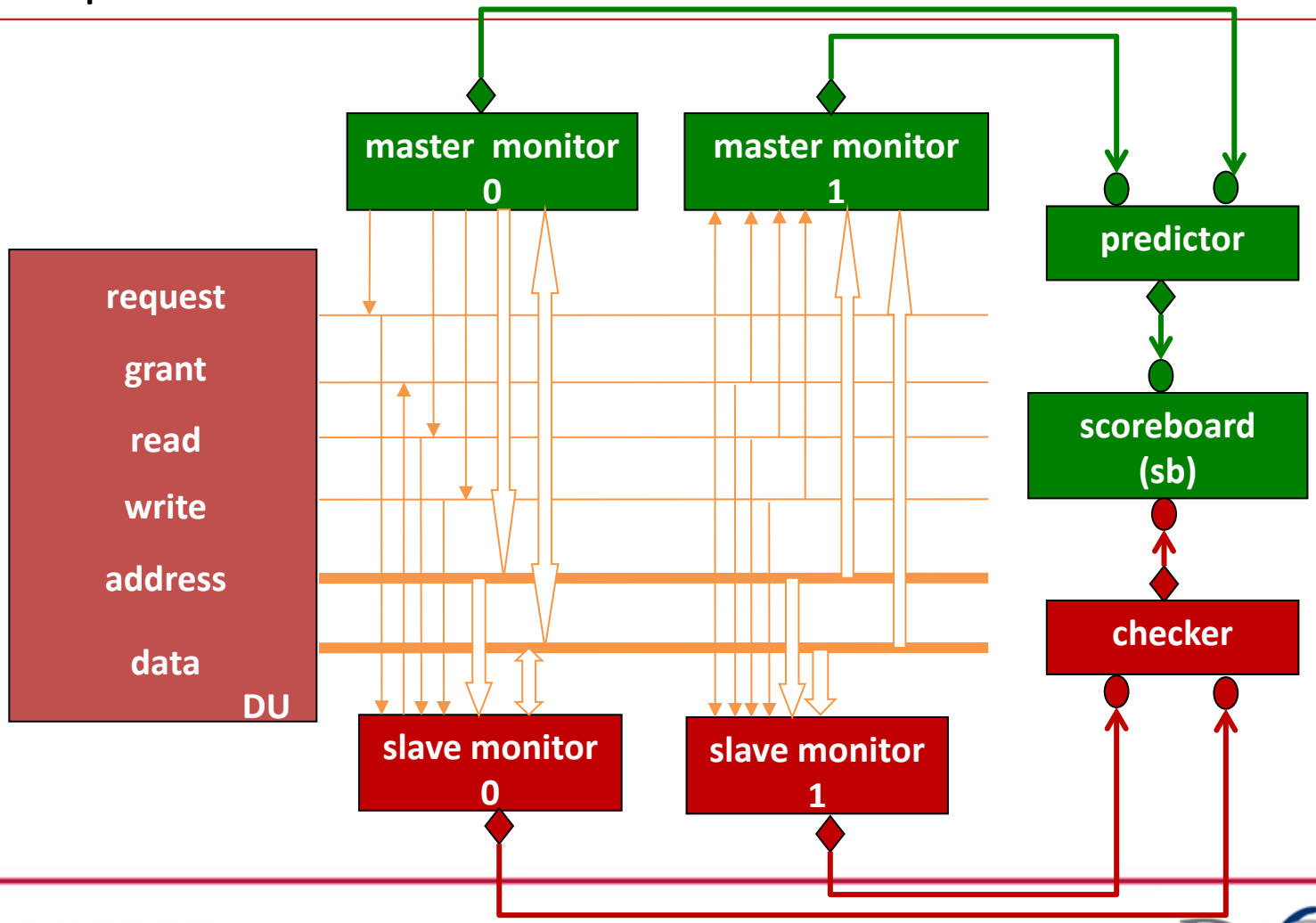
```
// Override
class acme_pw_scoreboard extends pw_scoreboard;

   // Identify the stream this transaction belongs to
   virtual function int get_stream_id(uvm_transaction t);
        acme_transfer xtr;
        int stream_id;

        // Get the stream id
        $cast(xtr, t);
        stream_id = calculate_stream(xtr.addr);
        return stream_id;
   endfunction: get_stream_id
```

Multiple Streams – Approach 2

- Use case example:
  - Multiple independent streams
  - In-order within stream
  - Out-of-order in between
  - Stream based on port id
- Steps
  - Add predictor/checker

XBUS Example

Multiple Masters and Slaves



XBUS Example

```
// Build
sb = xbus_pw_scoreboard #(xbus_transfer, xbus_transfer)::type_id::create("sb", this);
predict = xbus_predictor::type_id::create(“predict”, this);
check    = xbus_checker::type_id::create(“check”, this);

// Connect monitors to predictor
xbus0.masters[0].monitor.item_collected_port.connect(predict.inp_exports[0]);
xbus0.masters[1].monitor.item_collected_port.connect(predict.inp_exports[1]);

// Conenct monitors to checker
xbus0.slaves[0].monitor.pw_item_collected_port.connect(check.inp_exports[0]);
xbus0.slaves[1].monitor.pw_item_collected_port.connect(check.inp_exports[1]);

// Connect predictor/checker to scoreboard
predict.sb_aports[0].connect(sb.post_export);
check.sb_aports[0].connect(sb.check_export);
```

XBUS Example

```
// Predictor class
class xbus_predictor
        extends pw_predictor_checker (2,1, xbus_transfer, xbus_transfer);
  …
  // Override transfer task
  virtual task transfer(xbus_transfer trans, int port_id);
     // Post to scoreboard stream port_id
     sb.post_sb_data(trans, port_id);
  endtask : transfer
 …
// Checker class
class xbus_checker
        extends pw_predictor_checker (2,1, xbus_transfer, xbus_transfer);
  …
  // Override transfer task
  virtual task transfer(xbus_transfer trans, int port_id);
     // Check in scoreboard stream port_id
     sb.check_sb_data(trans, port_id);
 endtask : transfer
```

End-of-test Objections

- Use case example:
  - Raise objection when any outstanding checks
  - Lower objection when none left
  - Integrate with any end-of-test objection scheme
- Steps
  - Override
    - pw_scoreboard::raise_objection
    - pw_scoreboard::lower_objection

End-of-test Objections

```
// Override
class acme_pw_scoreboard extends pw_scoreboard;

   // Called whenever the scoreboard goes from empty to non-empty
   virtual function void raise_objection();
      // Application specific mechanism
      sm.raise(this.name);
   endfunction: raise_objection

   // Called whenever the scoreboard goes from non-empty to empty
   virtual function void lower_objection();
       // Application specific mechanism
       sm.lower(this.name);
   endfunction: lower_objection
```

Timeouts

- Use case example:
  - Error if posted item not checked before a certain event
- Steps
  - Extend pw_scoreboard::get_timeout()
    - Called automatically when the scoreboard sees something posted
    - Create and return an event variable to scoreboard
    - Implement the trigger condition on that variable

Timeouts

```
class acme_pw_scoreboard extends pw_scoreboard;
   …

   // This method is invoked when the transaction posted is pushed to the scoreboard
   // Return an event that will be triggered after 100ns
   virtual function uvm_event get_timeout(uvm_transaction posted);
      uvm_event    tmout_ev = new;
      fork
         begin
            #100ns;
            tmout_ev.trigger();
         end
      join_none
      return tmout_ev;
   endfunction:get_timeout
```

Summary

Summary

- Scoreboard is essential
- Proposed Implementation handles most use cases
  - Please let us know of unsupported use cases!
- Currently being used at several projects
  - http://downloads.paradigm-works.com/
  - 1050+ downloads to date from sourceforge
  - 128+ downloads of the UVM version
  - Incorporated several rounds of user feedback
  - Sourceforge bug-tracker/mantis maintained
  - pw-support@paradigm-works.com
- Use it freely!
  - Apache 2.0