UVM Analysis Port Functionality and Using Transaction Copy Commands

Clifford E. Cummings
Sunburst Design, Inc.

Heath Chambers
HMC Design Verification, Inc.

World-class SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!

October 23, 2018
Austin

Agenda

Basic queues, mailboxes and TLM FIFOs ← **1st pass**

Subscriber satellite TV analogy

Analysis paths & analysis ports, exports, and imps

TLM FIFOs ← **More detail**

Importance of the `copy()` method

How analysis port connections work - `write()` method

Summary & Conclusions

The paper has more details and more examples

Important SystemVerilog Features

PAY ATTENTION !!

- Queues
  - **`push_back()`** method to put a handle into the queue
  - **`foreach()`** method to walk through all stored handles
  - Does not have blocking **`get()`** method
- Mailboxes
  - Has nonblocking **`try_put()`** method
  - Has blocking **`get()`** method
- Analysis path considerations:
  - Must start with **`uvm_analysis_port`** and end with **`uvm_analysis_imp`**
  - **`uvm_tlm_fifo`** ***cannot*** terminate an analysis path
  - **`uvm_tlm_analysis_fifo`** ***CAN*** terminate an analysis path

Queues will be used to store *component* handles

Can store class handles - *great for storing connected components*

Not too useful for scoreboards

Mailboxes will be used to store *transaction* handles

Can store class handles - *great for storing transactions*

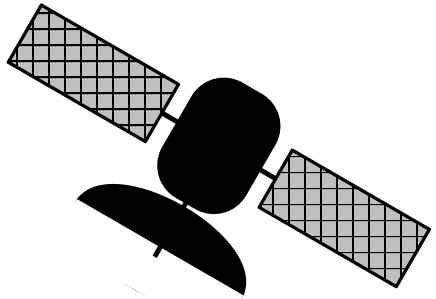
Important for scoreboards

Must include `write()` method

Built using mailboxes

Very useful for scoreboards !!

Subscriber Satellite TV Analogy



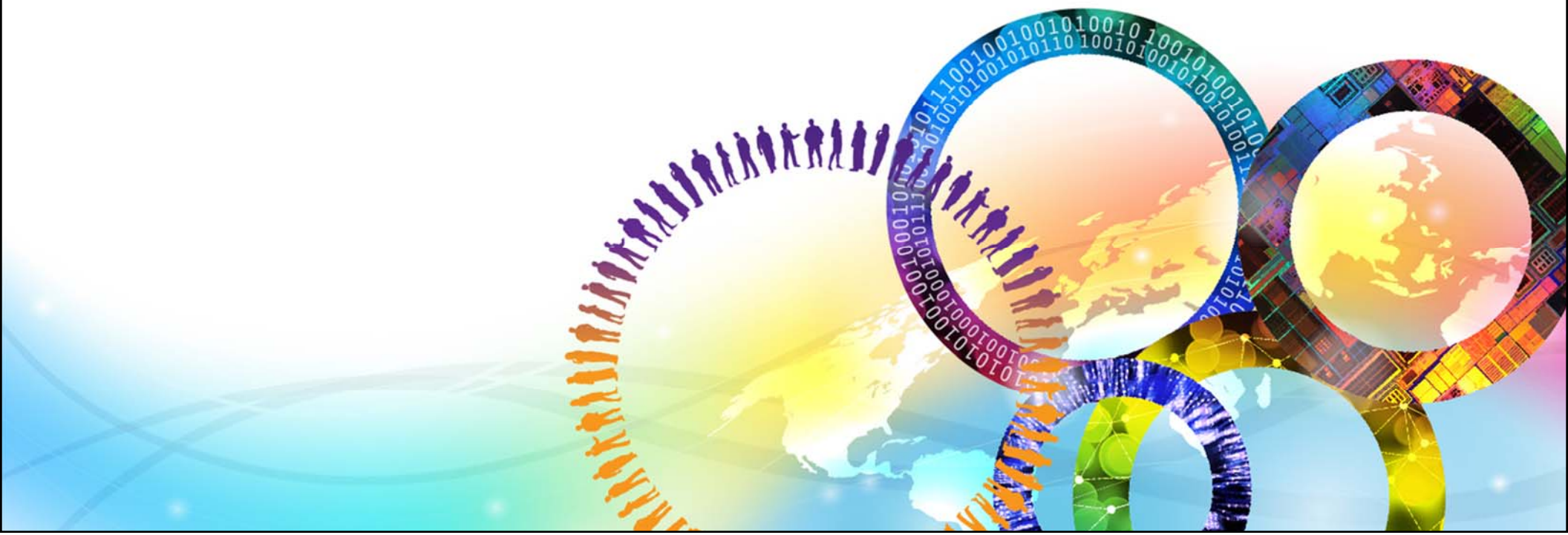
- Two ways to watch a broadcast satellite TV program
  - Watch the program live
  - Record the program to a DVR to view later
- Satellite programs are broadcast as scheduled
  - **There might be 1,000's of viewers**
  - **There might be *NO* viewers**
- No way to restart a broadcast program
  - **No way to communicate back to the satellite**
  - **Other viewers would object to restarting the program**
- Subscribers not allowed to change the live program
  - **With the right equipment, you can modify your copy**

Sunburst Design

snug
Synopsys Users Group
Austin

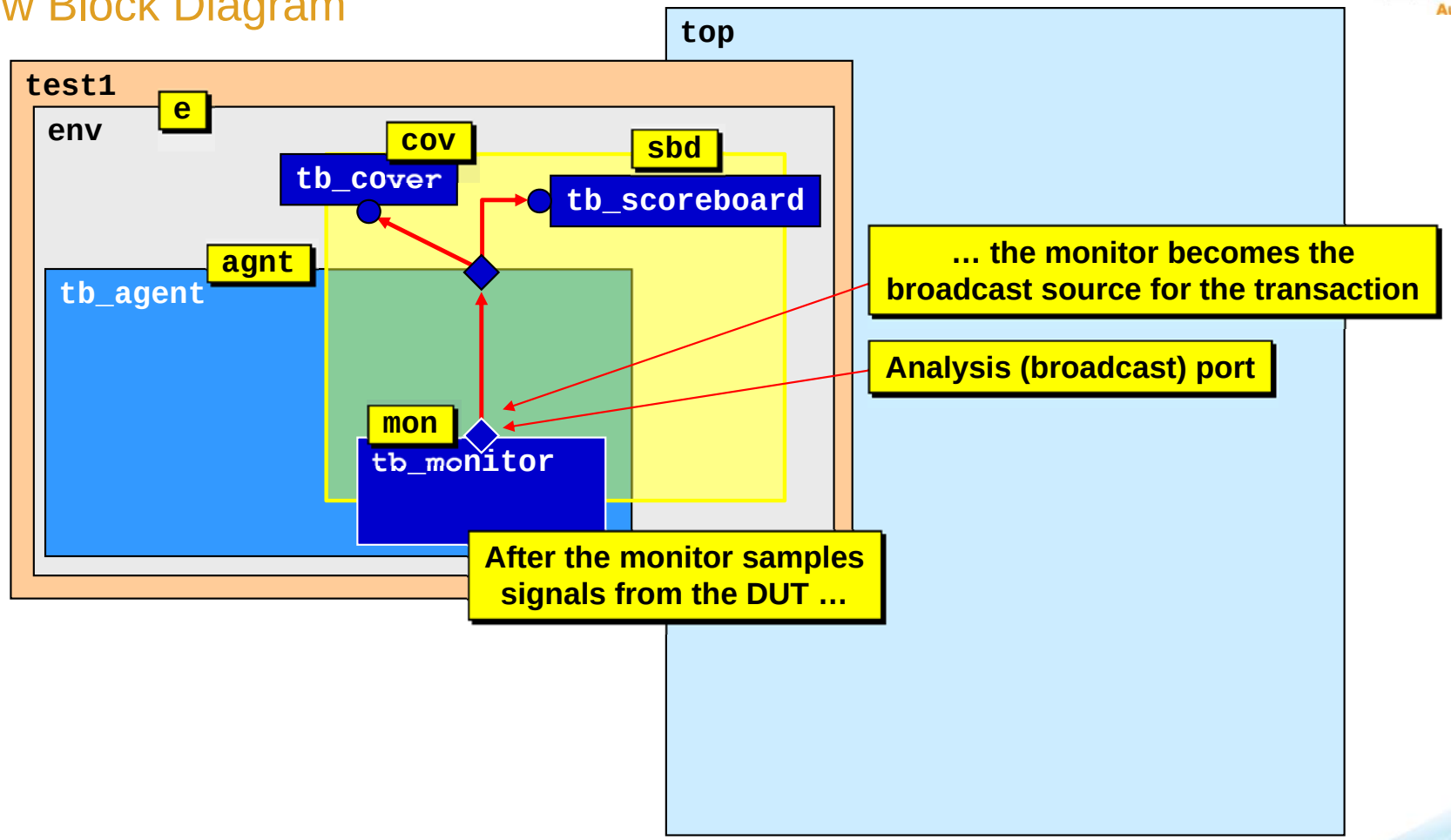
Analysis Port Connections

and TLM FIFOs



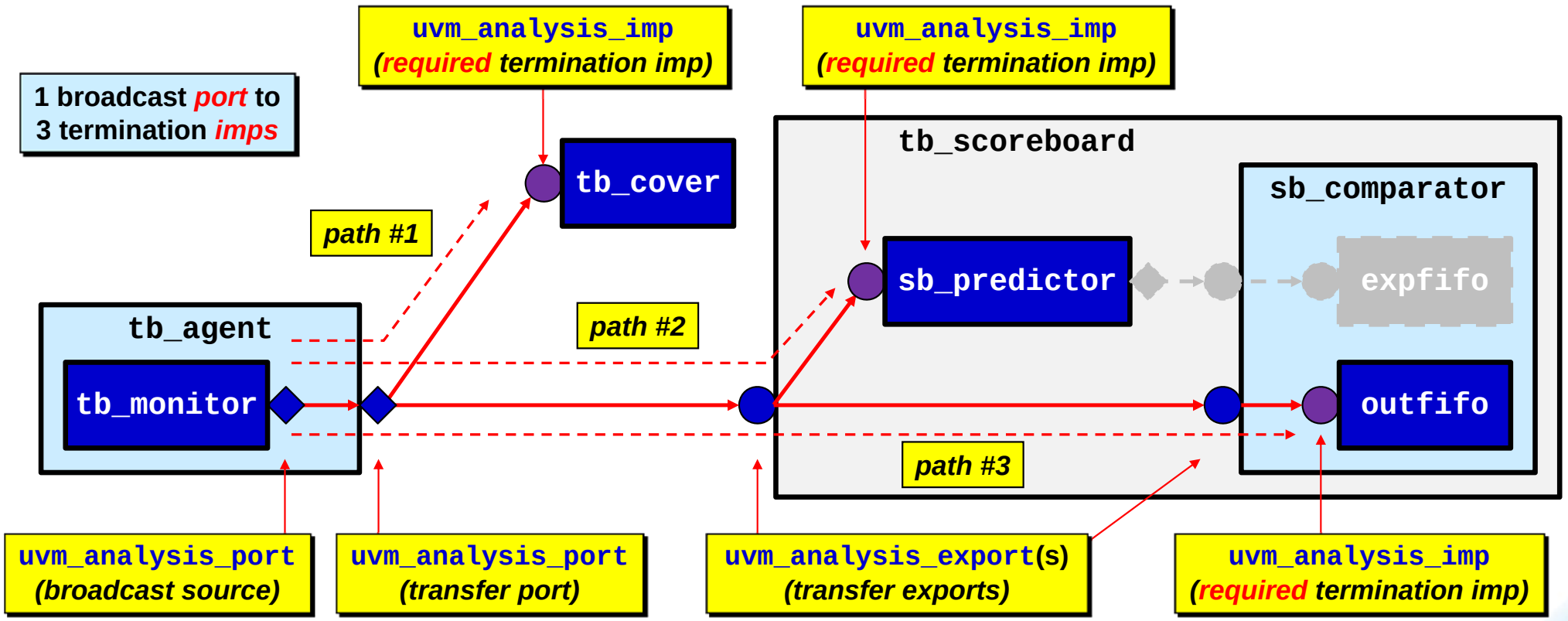
Common UVM Components

Overview Block Diagram



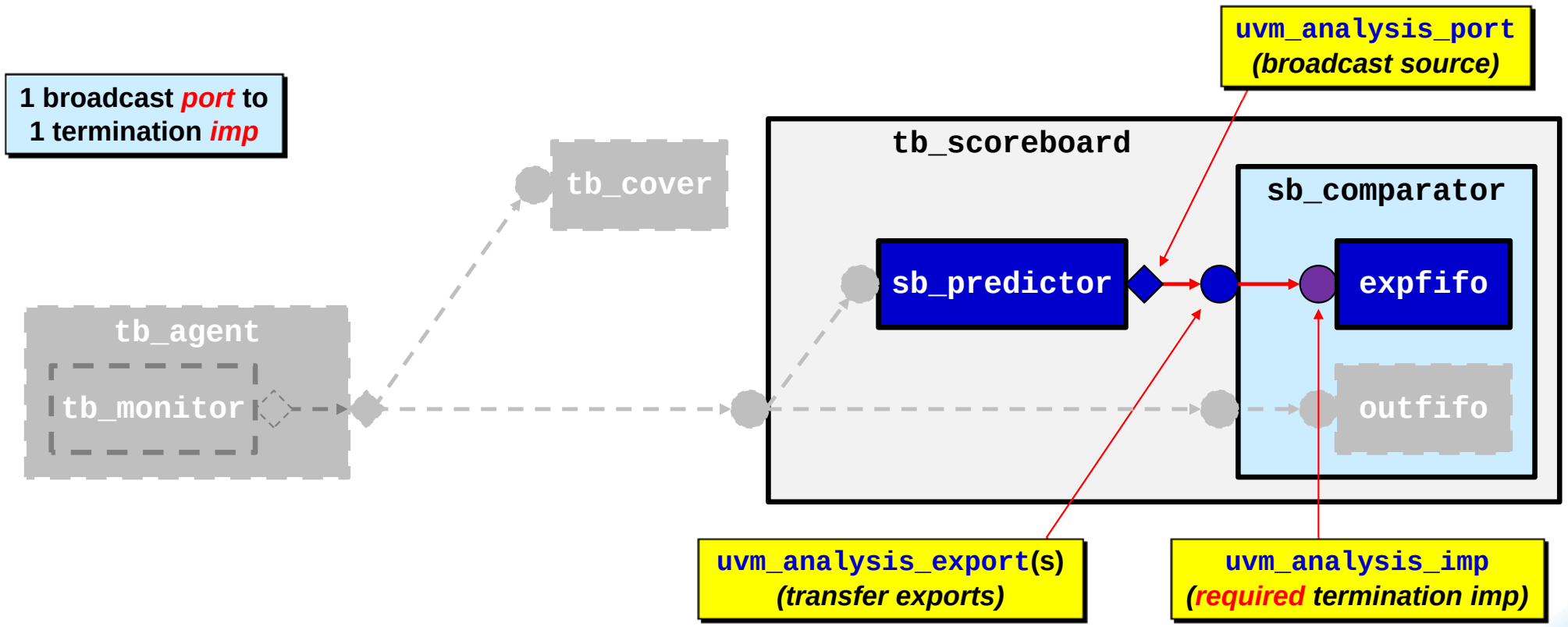
UVM Testbench Analysis Port Paths

Common Paths - Monitor to Multiple Subscribers



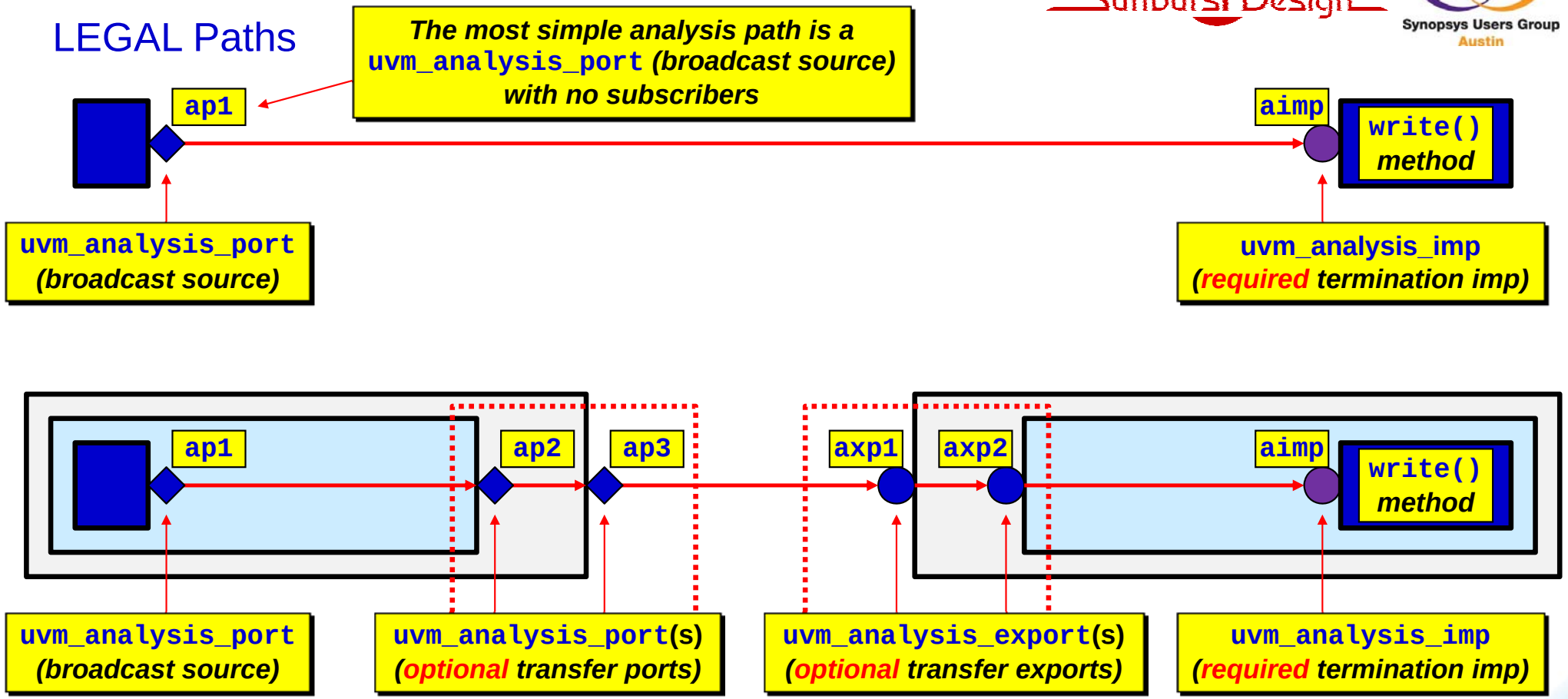
UVM Testbench Analysis Port Paths

Common Paths - Predictor to Expected Transaction FIFO



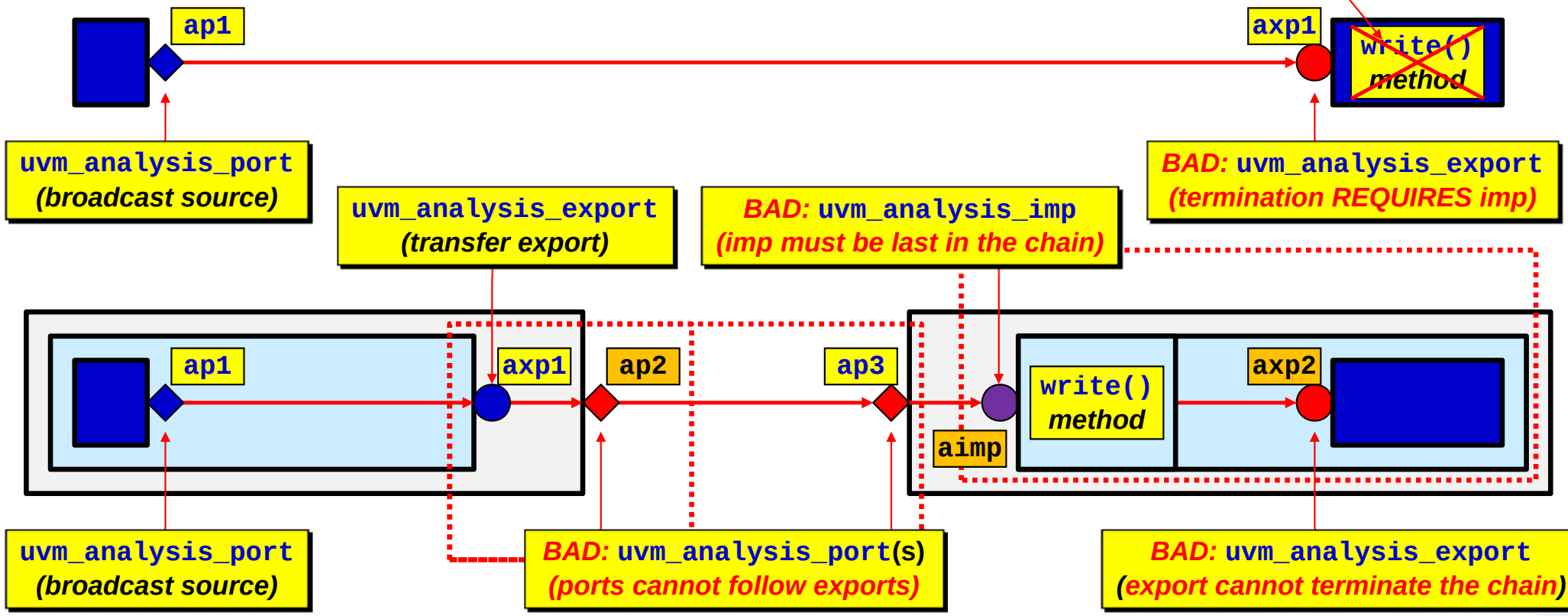
UVM Analysis Port Paths

LEGAL Paths



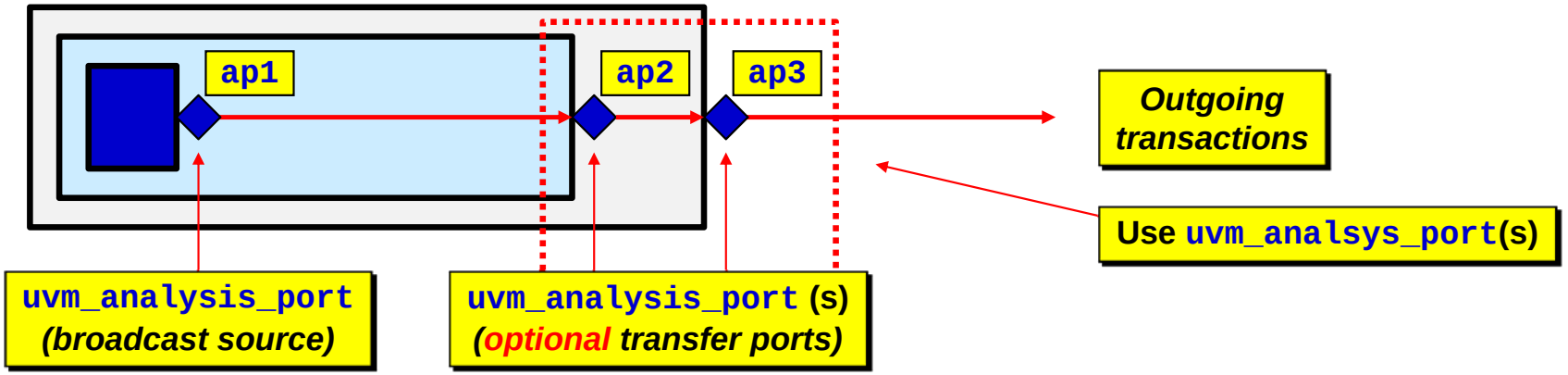
UVM Analysis Port Paths

ILLEGAL Paths



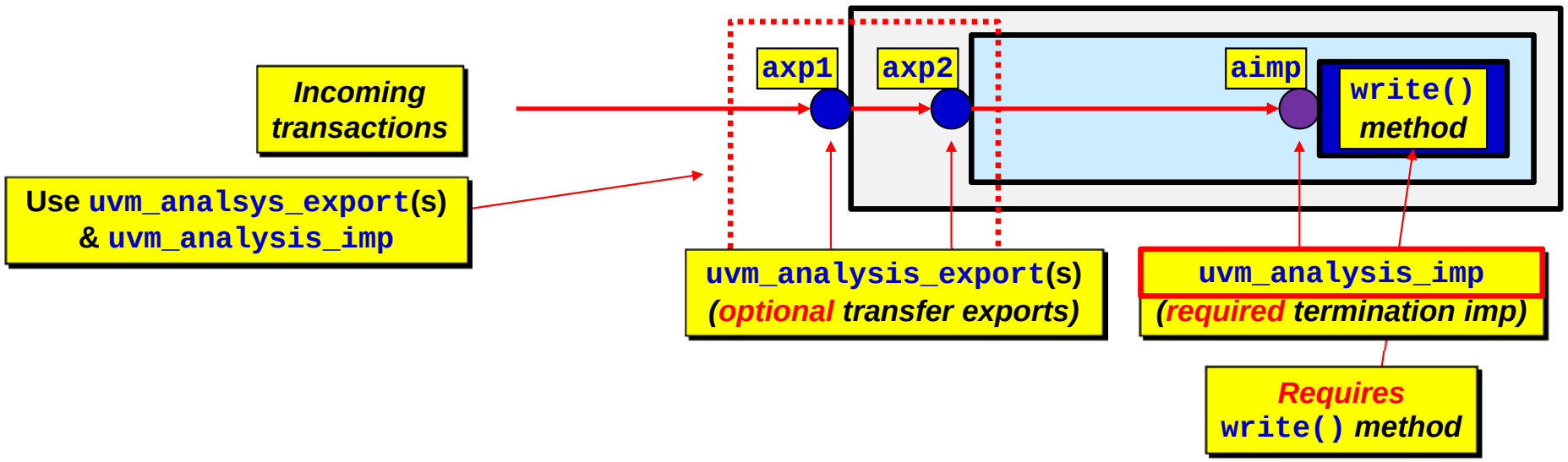
UVM Analysis Ports

Recommended Usage



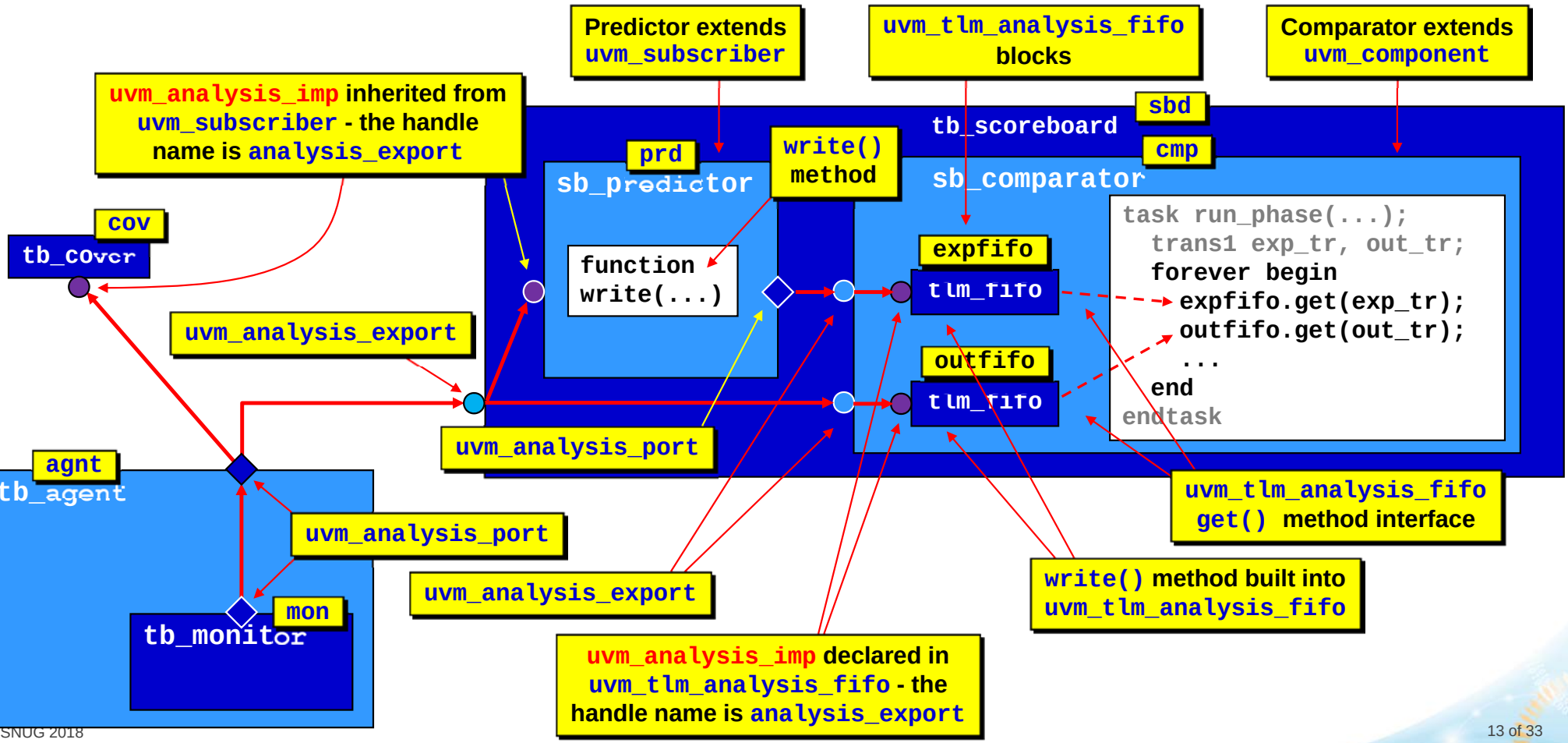
UVM Analysis Exports & Imps

Recommended Usage



Common Analysis Port Connections

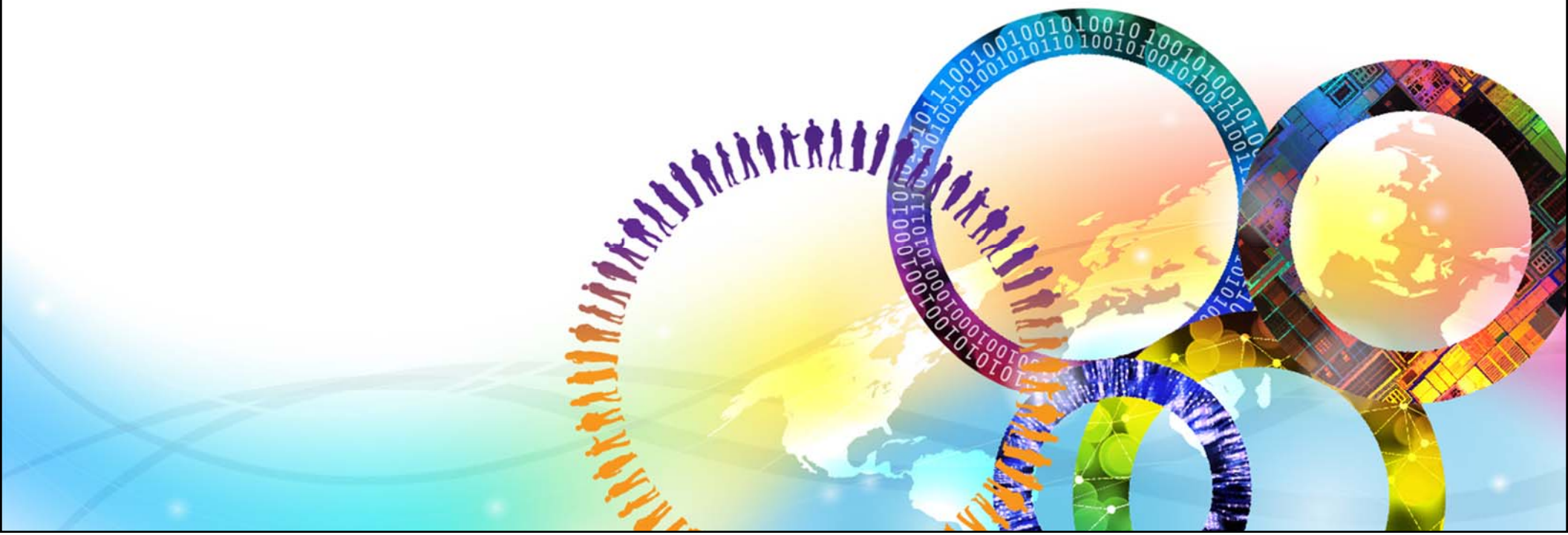
Recommended Connections



Sunburst Design

snug
Synopsys Users Group
Austin

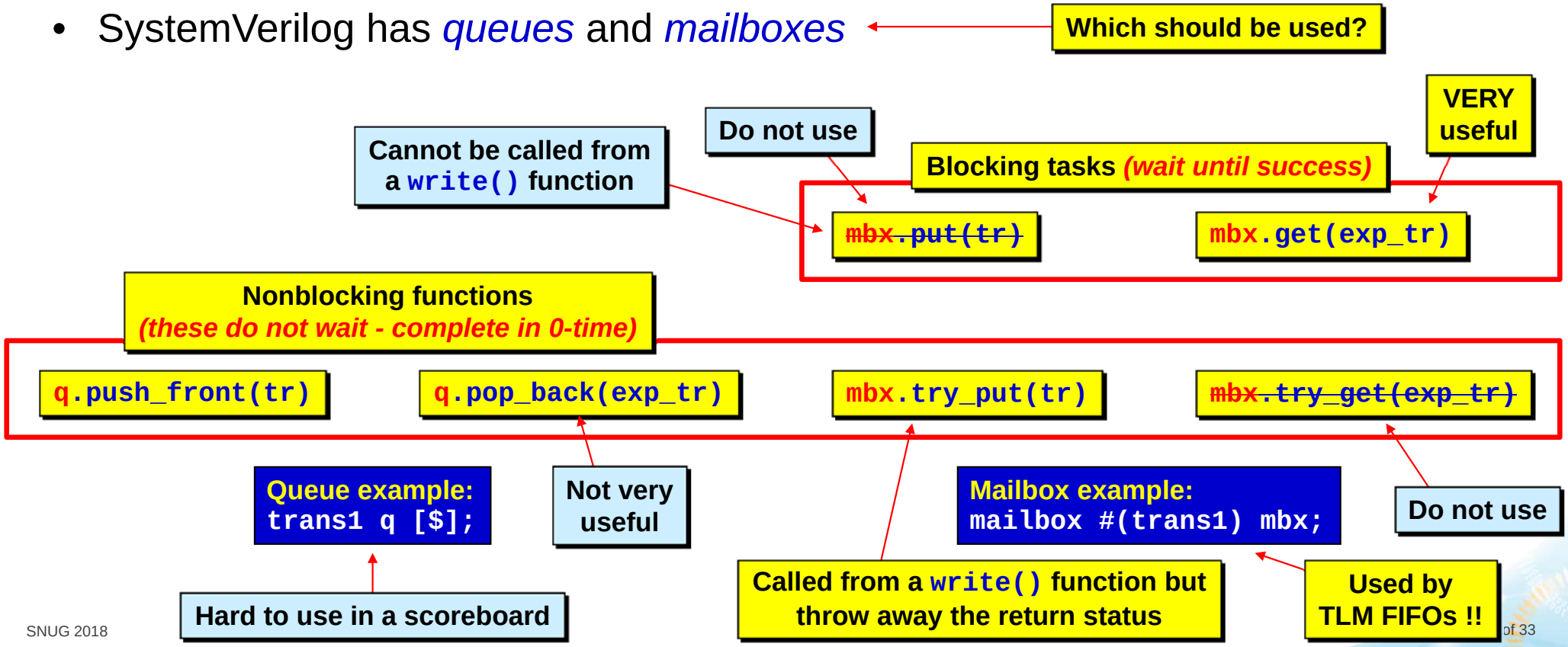
TLM FIFOs - Definitions & Usage



TLM FIFOs & Scoreboards

SystemVerilog Queues & Mailboxes

- Scoreboards typically store *expected* and *actual* transactions
- SystemVerilog has *queues* and *mailboxes*



uvm_tlm_fifo

Most Common Usage

unburst Design

snug
Synopsys Users Group
Austin

Although there are 2 *non-analysis* `imp` ports and 2 `analysis` ports on the `uvm_tlm_fifo`, they typically are not used

The `uvm_tlm_fifo`, will be constructed to be unbounded. Example:
`exp_fifo=new("exp_fifo", this, 0);`

`0` means unbounded

`try_put()` method stores into the mailbox

`uvm_tlm_fifo`
`local mailbox #(T) m`

`get()` method retrieves from the mailbox

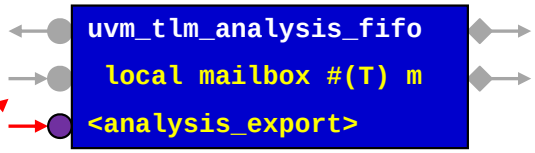
The `uvm_analysis_imp` `write(tr)` method will call `void'(try_put(tr))`

The scoreboard comparator will call the blocking `get(tr)` method and wait to retrieve a `uvm_tlm_fifo` transaction

`void`-cast to throw away the `try_put()` return-status
(`try_put()` always succeeds on unbounded fifo's mailbox)

uvm_tlm_analysis_fifo

Most Common Usage



unburst Design

snug
Synopsys Users Group
Austin

Although there are 2 *non-analysis* `imp` ports and 2 `analysis` ports on the `uvm_tlm_fifo`, they typically are not used

The `uvm_tlm_analysis_fifo` is unbounded by default

Termination of an analysis-path

`uvm_tlm_analysis_fifo`

`local mailbox #(T) m`

`get()` method retrieves from the mailbox

`uvm_analysis_imp` with handle name `analysis_export` is almost always used

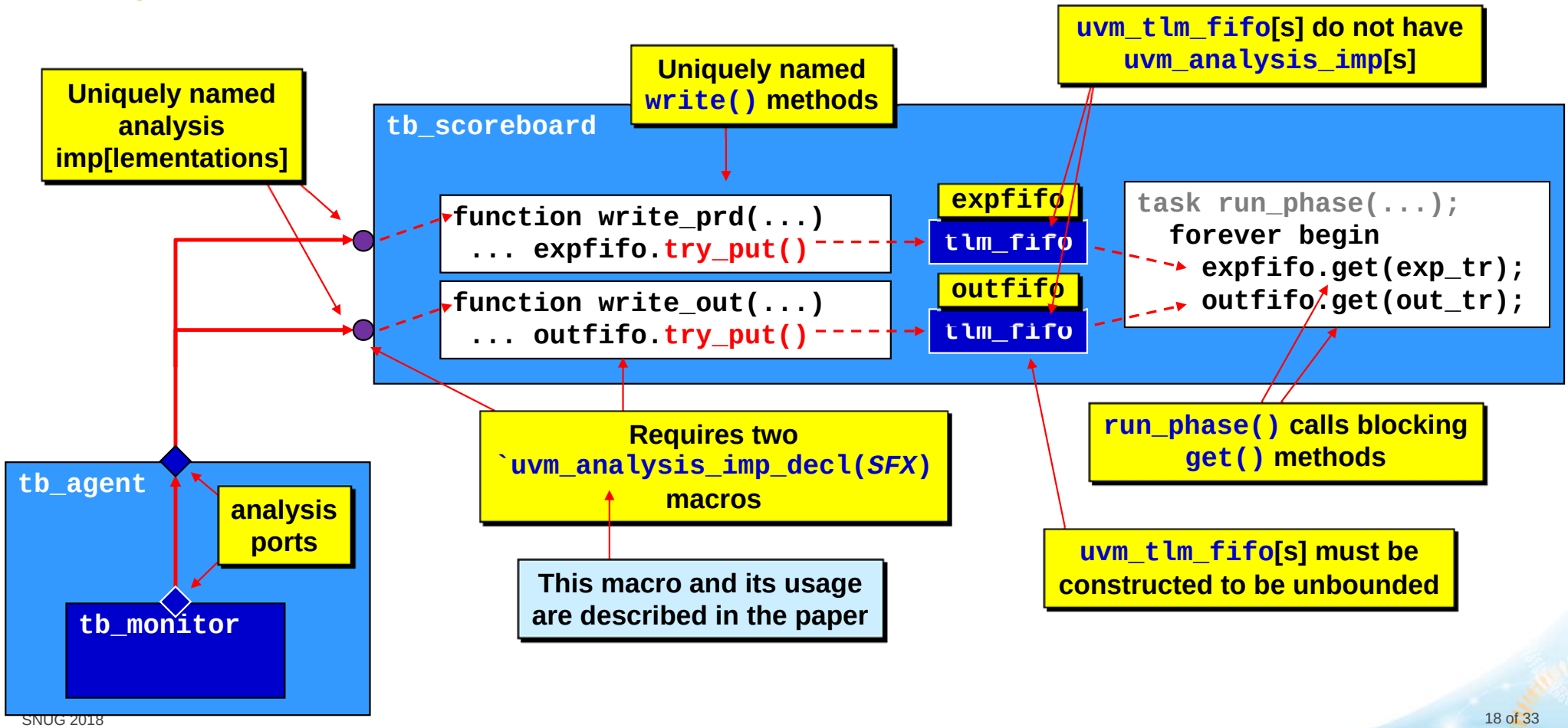
`imp` already has `write(t)` method built-in

Internally executes: `void'(this.try_put(t));`

The scoreboard comparator still calls blocking `get(tr)` method and waits to retrieve a `uvm_tlm_analysis_fifo` transaction

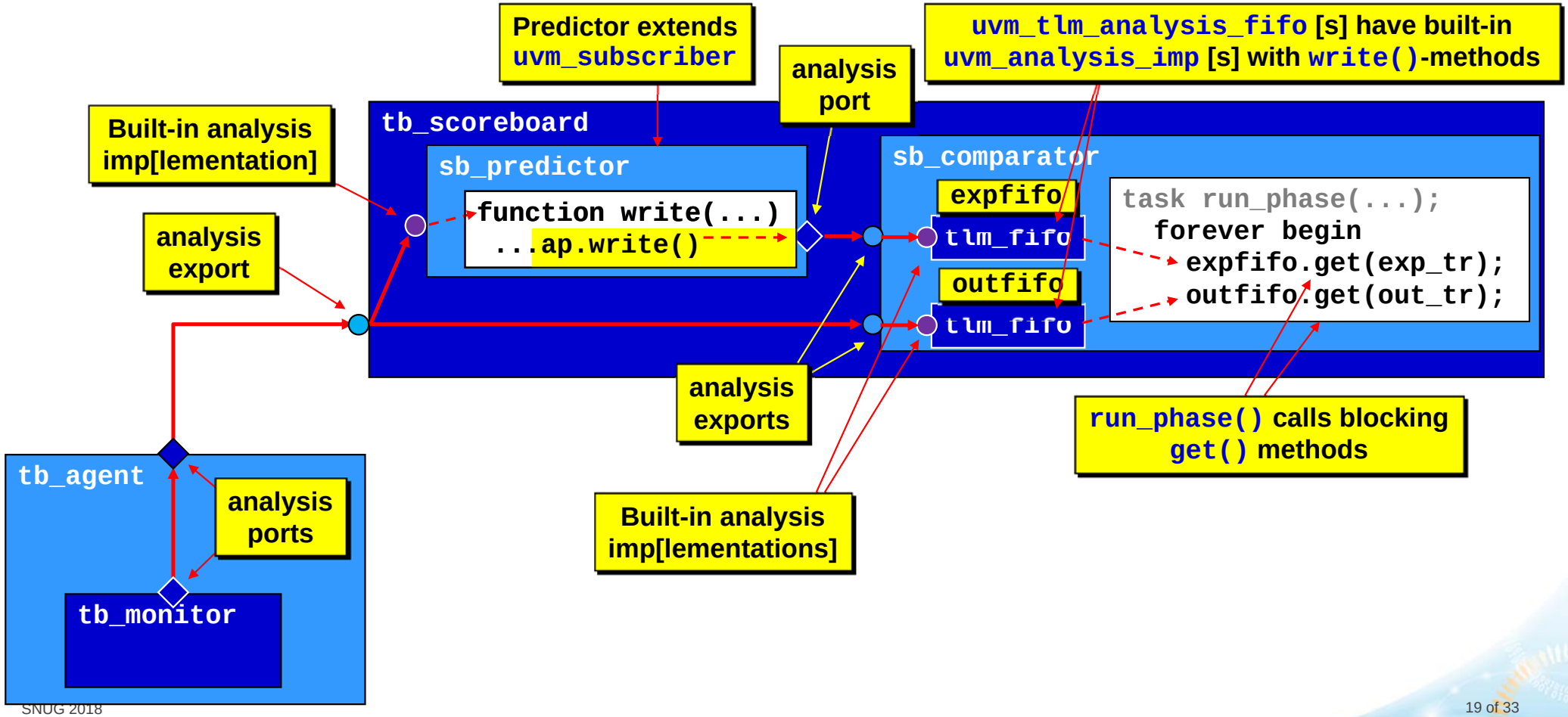
Typical Scoreboard

Using uvm_tlm_fifos

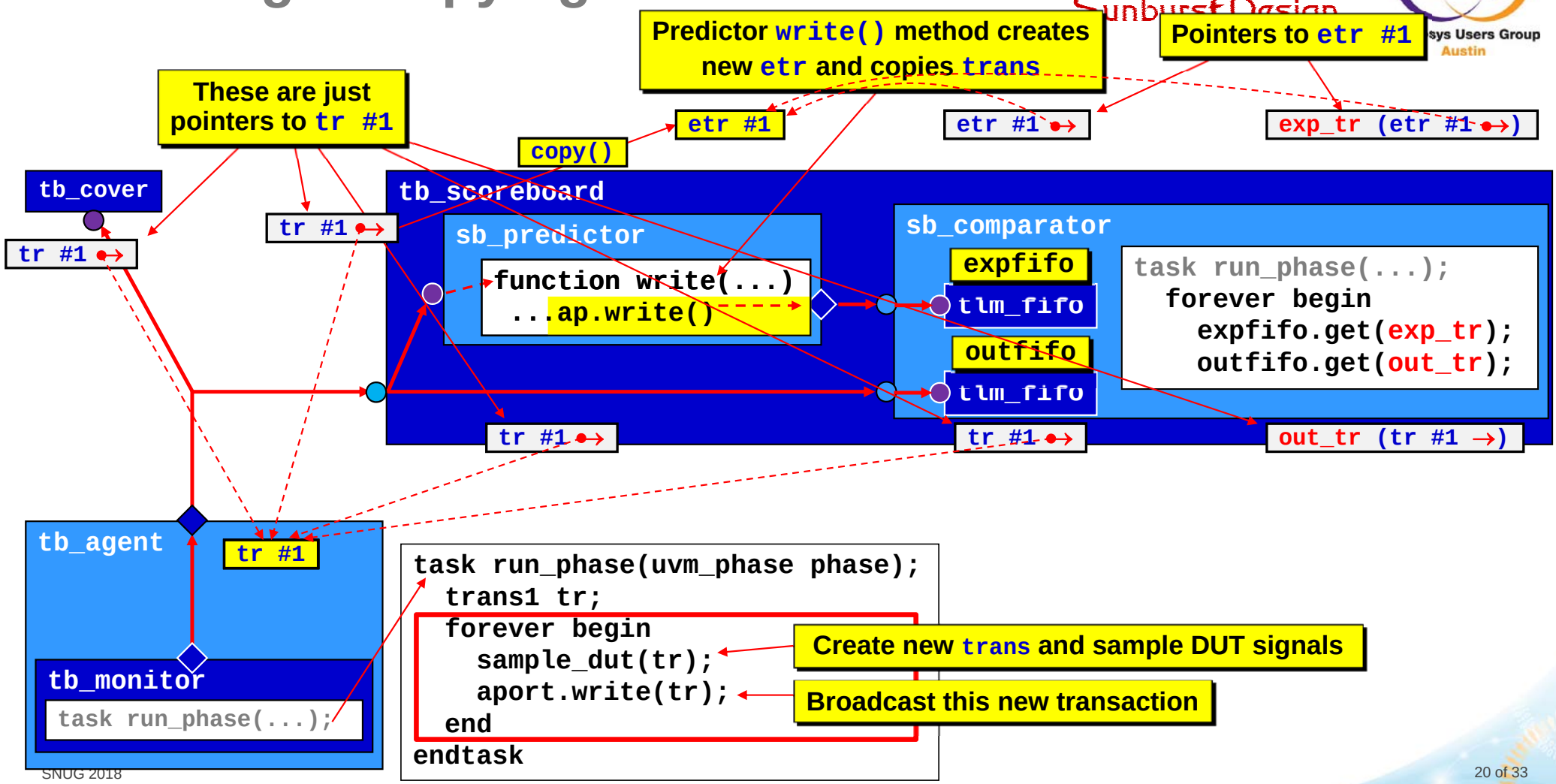


Typical Scoreboard

Using uvm_tlm_analysis_fifos

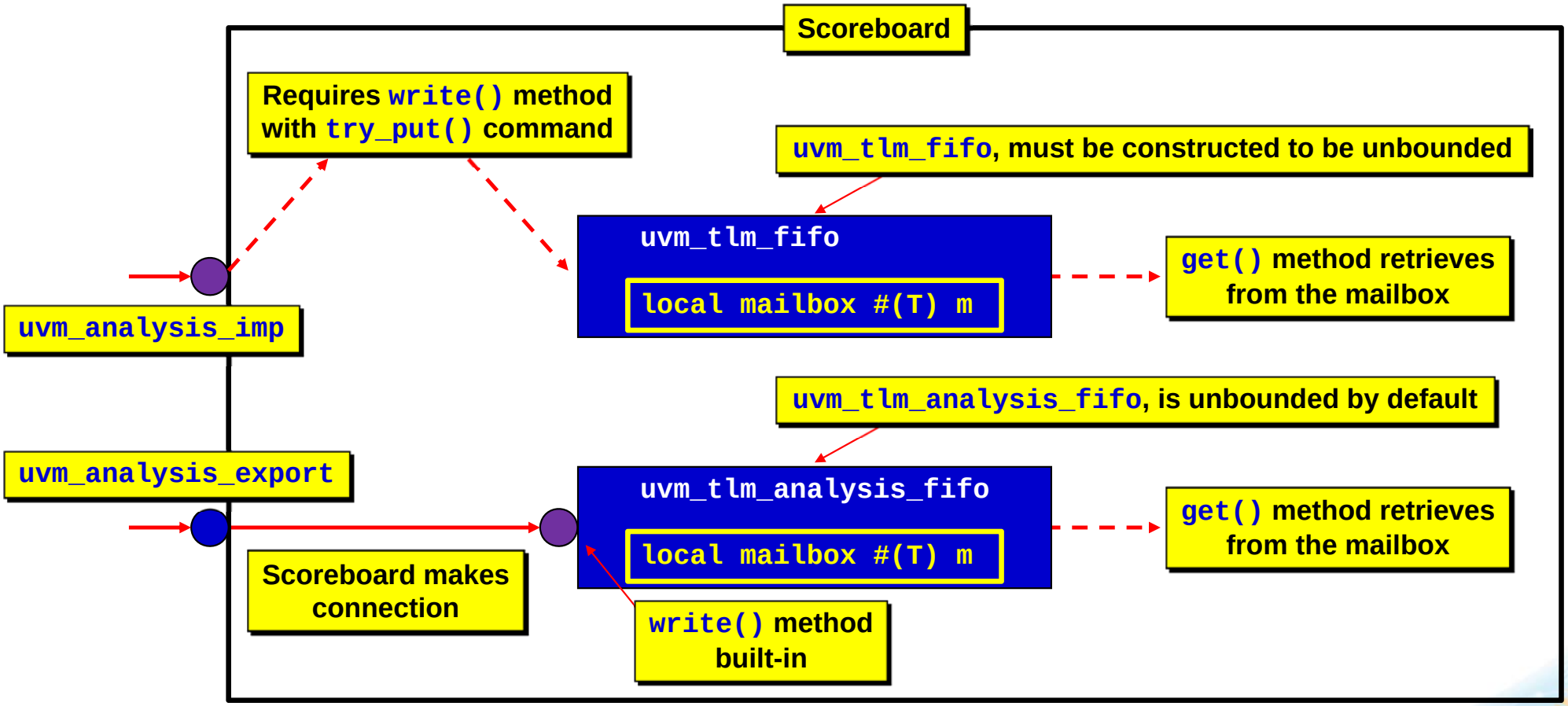


Creating & Copying Transactions



Comparing TLM FIFOs

uvm_tlm_fifo -vs- uvm_tlm_analysis_fifo



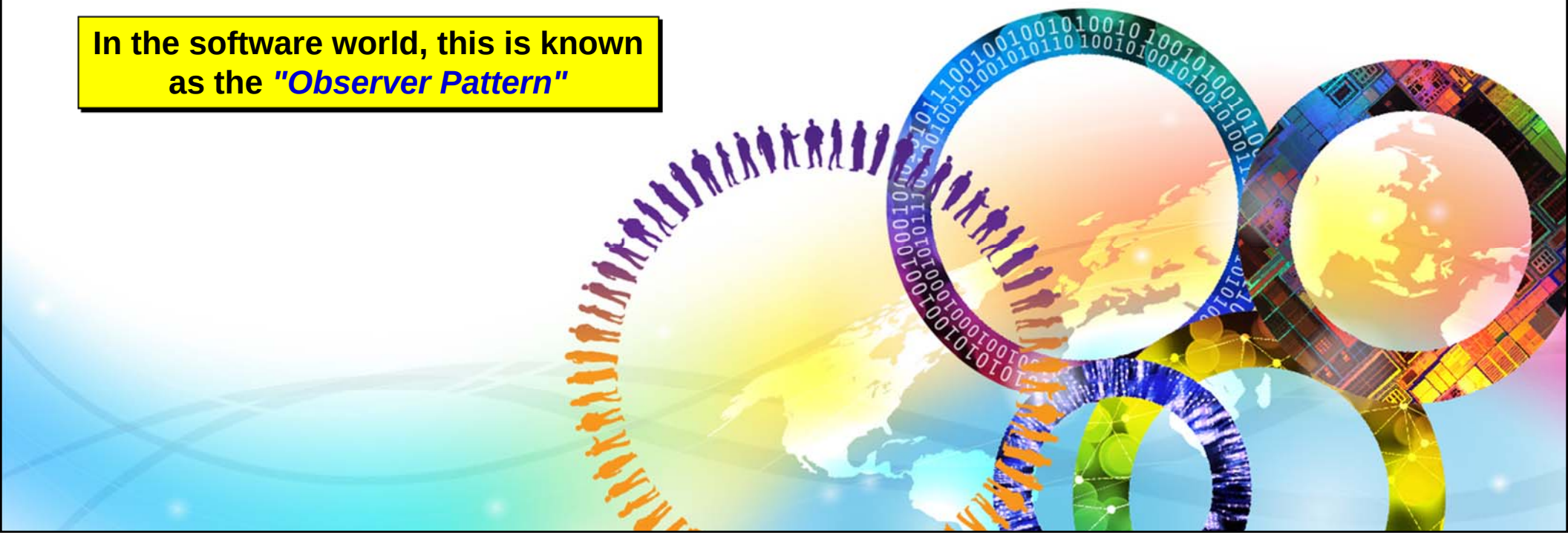
Sunburst Design

snug
Synopsys Users Group
Austin

Analysis Path Basics

How do analysis port-paths work?

In the software world, this is known as the *"Observer Pattern"*



How Does UVM Work?

On previous slides

- We have learned about analysis ports & TLM FIFOs

You now know how to use:
`uvm_analysis_port`
`uvm_analysis_export`
`uvm_analysis_imp`
`uvm_tlm_fifo`
`uvm_tlm_analysis_fifo`

- You do not have to know how UVM works

You now have enough knowledge to use analysis components

- The best engineers want to have *some* understanding on how UVM works
- The remaining slides show how UVM makes subscribers work

This is *NOT* UVM code !!

This is a *basic* version of what UVM does internally

These slides show how UVM uses *queues* and `foreach` loops to call each subscriber's `write()` method

This is a high-level tutorial on how monitors and subscribers work

This is not exactly how UVM works, *but it is close*

Monitor with Multiple Subscribers

Goal

- Create a *Monitor* that can connect to any number of subscribers and can call a `write()` method from each subscriber ***without modifying the Monitor code***

– **Version #1** ← **top module must know subscriber handle names in the Monitor** ✘

The monitor …

Must declare each subscriber handle ✘ **Has no `connect()` method Must copy handles by name** ✘ **Must call `write()` method for each subscriber** ✘

– **Version #2** ← **Monitor w/ generic `connect()` method to hide subscriber handle names** ✓ **UVM Like!**

The monitor …

Has queue of subscriber handles ✓ **Defines common `connect()` method for all subscribers** ✓ **Uses `foreach` loop to call `write()` methods using queued subscriber handles** ✓

Monitor & Subscribers

Version 1 - No connect() method

virtual analysis_if base class

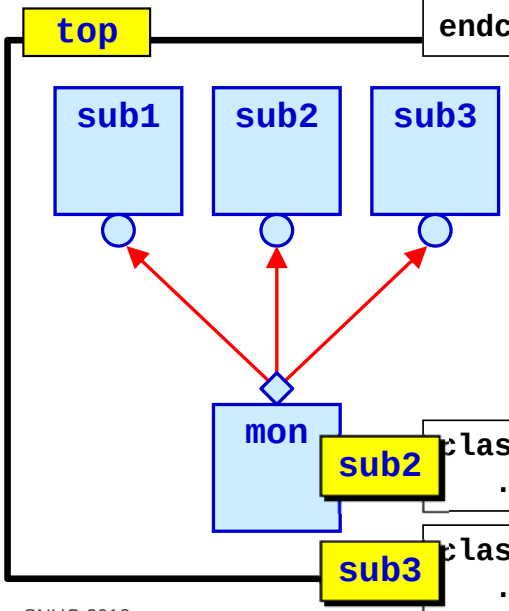
Extended classes must implement write() method

```
virtual class analysis_if;
  pure virtual task write(trans1 t);
endclass
```

```
class subscriber1 extends analysis_if;
  virtual task write(trans1 t);
    $display("subscriber1: ",
       "received ...", ...);
  endtask
endclass
```

sub1

top



sub2

sub3

Each subscriber handle is copied to the ap1-3 handles in monitor1

```
...
mon.ap1 = sub1;
mon.ap2 = sub2;
mon.ap3 = sub3;
...
```

In top module

```
class subscriber2 extends analysis_if;
  ... virtual task write(...) ...
```

```
class subscriber3 extends analysis_if;
  ... virtual task write(...) ...
```

```
class monitor1;
  analysis_if ap1;
  analysis_if ap2;
  analysis_if ap3;

  task run();
    trans1 t = new();
    repeat(5) begin
      void'(t.randomize());
      $display("monitor:   ",
       "**BROADCAST** ...", ...);
      ap1.write(t);
      ap2.write(t);
      ap3.write(t);
    end
  endtask
endclass
```

subscriber1 sub1 to ap1

subscriber2 sub2 to ap2

subscriber3 sub3 to ap3

Monitor & Subscribers

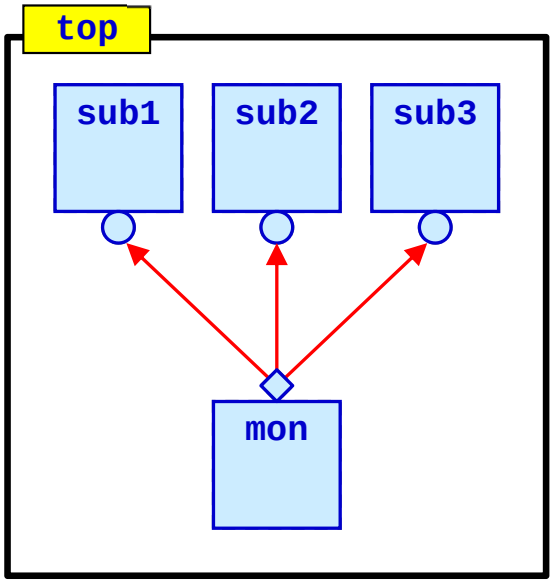
Version 1 - No connect() method

Declare `monitor1` and `subscriber1-3` handles

`new()`-construct `mon` and `sub1-3`

Copy `sub1-3` handles to `ap1-3` handles in `monitor1`

Call the `mon.run()` task




```
module top;
  import tb_pkg::*;

  monitor1    mon;
  subscriber1 sub1;
  subscriber2 sub2;
  subscriber3 sub3;

  initial begin
    mon  = new();
    sub1 = new();
    sub2 = new();
    sub3 = new();
    mon.ap1 = sub1;
    mon.ap2 = sub2;
    mon.ap3 = sub3;
    mon.run();
  end
endmodule
```

```
class monitor1;
  analysis_if ap1;
  analysis_if ap2;
  analysis_if ap3;

  task run();
    trans1 t = new();
    repeat(5) begin
      void'(t.randomize());
      $display("monitor:   ",
       "**BROADCAST** ...", ...);
      ap1.write(t);
      ap2.write(t);
      ap3.write(t);
    end
  endtask
endclass
```

Monitor must declare each `analysis_if`

With no `connect()` method in `monitor1`, the `top` module must reference names declared in `monitor1`

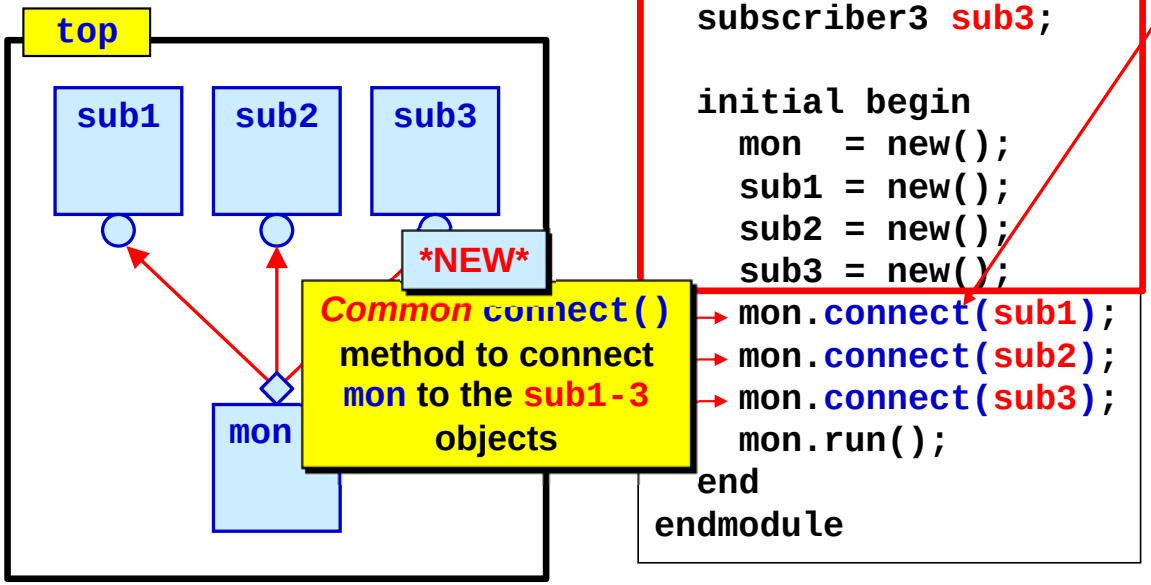
Repeat 5 times

`randomize()` transaction

Separately call each `ap[#].write()` method

Monitor & Subscribers

Version 2 - Adds analysis_if queue



No change from Version 1

```
module top;
  import tb_pkg::*;

  monitor2    mon;
  subscriber1 sub1;
  subscriber2 sub2;
  subscriber3 sub3;

  initial begin
    mon  = new();
    sub1 = new();
    sub2 = new();
    sub3 = new();
    mon.connect(sub1);
    mon.connect(sub2);
    mon.connect(sub3);
    mon.run();
  end
endmodule
```

NEW

Monitor declares queue of analysis_if ports

NEW

Each call to connect() method will push_back another analysis_if onto the ap-queue

Common connect() method

Each subscriber's write() method is called from the ap-queue

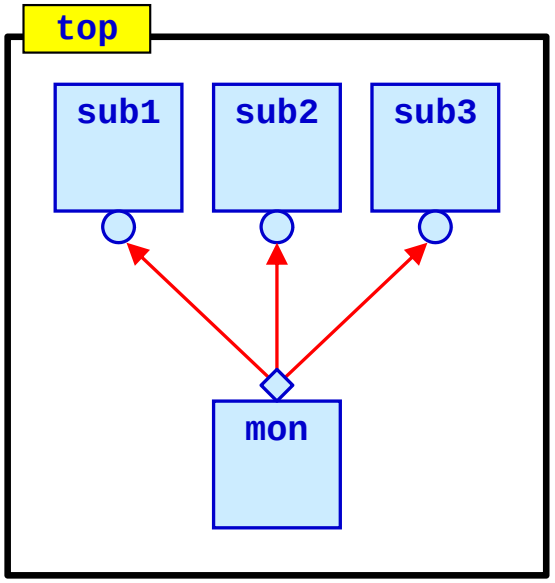
```
class monitor2;
  analysis_if ap[$];

  function void connect (analysis_if port);
    ap.push_back(port);
  endfunction

  task run();
    trans1 t = new();
    repeat(5) begin
      void'(t.randomize());
      $display("monitor:    ",
        "***BROADCAST** ...", ...);
      foreach(ap[i]) ap[i].write(t);
    end
  endtask
endclass
```

NEW

More subscribers could be added to top module without modifying monitor2 code

Monitor & Subscribers

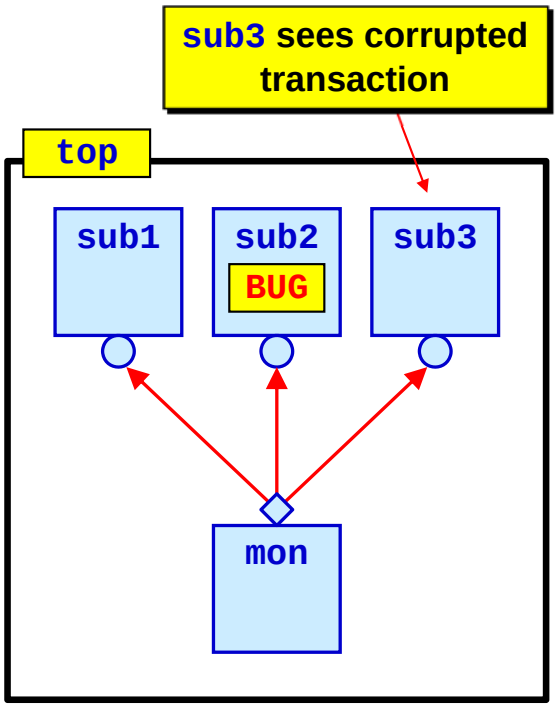
Simulation Output




```
Randomized trans1 values addr=f9  data=50
monitor:   **BROADCAST** addr=f9  data=50
subscriber1: received    addr=f9  data=50
subscriber2: received    addr=f9  data=50
subscriber3: received    addr=f9  data=50

Randomized trans1 values addr=e9  data=27
monitor:   **BROADCAST** addr=e9  data=27
subscriber1: received    addr=e9  data=27
subscriber2: received    addr=e9  data=27
subscriber3: received    addr=e9  data=27

...
```

Subscriber2 BUG

Version 4 - modifies transaction values




```
class subscriber1 extends analysis_if;
  virtual task write(trans1 t);
    $display("subscriber1: ", "received addr=%2h  data=%2h", t.addr, t.data);
  endtask
endclass
```

subscriber1 has the original transaction addr & data values

```
class subscriber2 extends analysis_if;
  virtual task write(trans1 t);
    $display("subscriber2: ", "received addr=%2h  data=%2h", t.addr, t.data);

`ifdef BUG
    t.addr = 8'hFF;
    t.data = 8'h00;
    $display("subscriber2: ", "set      addr=%2h  data=%2h", t.addr, t.data);
`endif

  endtask
endclass
```

BUG: subscriber2 modifies the addr & data of the broadcast transaction

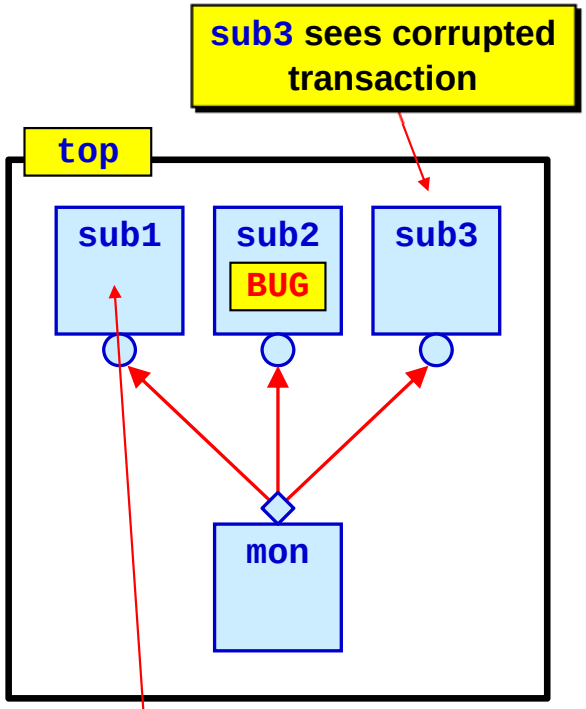
NEVER **modify the broadcast transaction !!**

```
class subscriber3 extends analysis_if;
  virtual task write(trans1 t);
    $display("subscriber3: ", "received addr=%2h  data=%2h", t.addr, t.data);
  endtask
endclass
```

subscriber3 now sees the modified transaction addr & data values

Monitor & Subscribers

BUG: Simulation Output




```
Randomized trans1 values addr=f9  data=50
monitor:   **BROADCAST** addr=f9  data=50
subscriber1: received    addr=f9  data=50
subscriber2: received    addr=f9  data=50
subscriber2: set         addr=ff  data=00
subscriber3: received    addr=ff  data=00

Randomized trans1 values addr=e9  data=27
monitor:   **BROADCAST** addr=e9  data=27
subscriber1: received    addr=e9  data=27
subscriber2: received    addr=e9  data=27
subscriber2: set         addr=ff  data=00
subscriber3: received    addr=ff  data=00

...
```

Depending on how the subscribers are pushed onto the ap - queue, sub1 might also see the bug

Transaction Copy() Method

- All subscribers receive a handle to the *same* broadcast transaction
- A subscriber should ***NEVER*** modify contents of the received transaction
- Any subscriber that modifies transaction contents ***MUST take a copy before making modifications***

Summary & Conclusions

- Analysis ports are ports that broadcast transactions to 0 or more destinations
- Each subscriber chain terminates with a **`uvm_analysis_imp`** and corresponding **`write()`** method
- Subscribers should ***NEVER*** modify the broadcast transaction
- Subscribers need to use the transaction in 0-time

  -OR-
- Subscribers need to take a local copy
- If a component has multiple **`imp`**-inputs, use the macro:

  `uvm_analysis_imp_decl(SFX)`

  This is described in the paper
- The **`uvm_tlm_analysis_fifo`** has a built-in **`uvm_analysis_imp`** port
- Prove that the scoreboard analysis paths are working

  Great feature for terminating an analysis path in a scoreboard

DO NOT ASSUME **that the analysis paths are working correctly !!**

Acknowledgements

Thanks!

- We are grateful to our colleagues Jeff Vance, Kelly Larson and Don Mills for their reviews and suggested improvements to our paper

- Thanks also to David Lee, Don Mills, Jeff Vance, Kelly Larson and Dan Chaplin for helping to improve the presentation content and flow

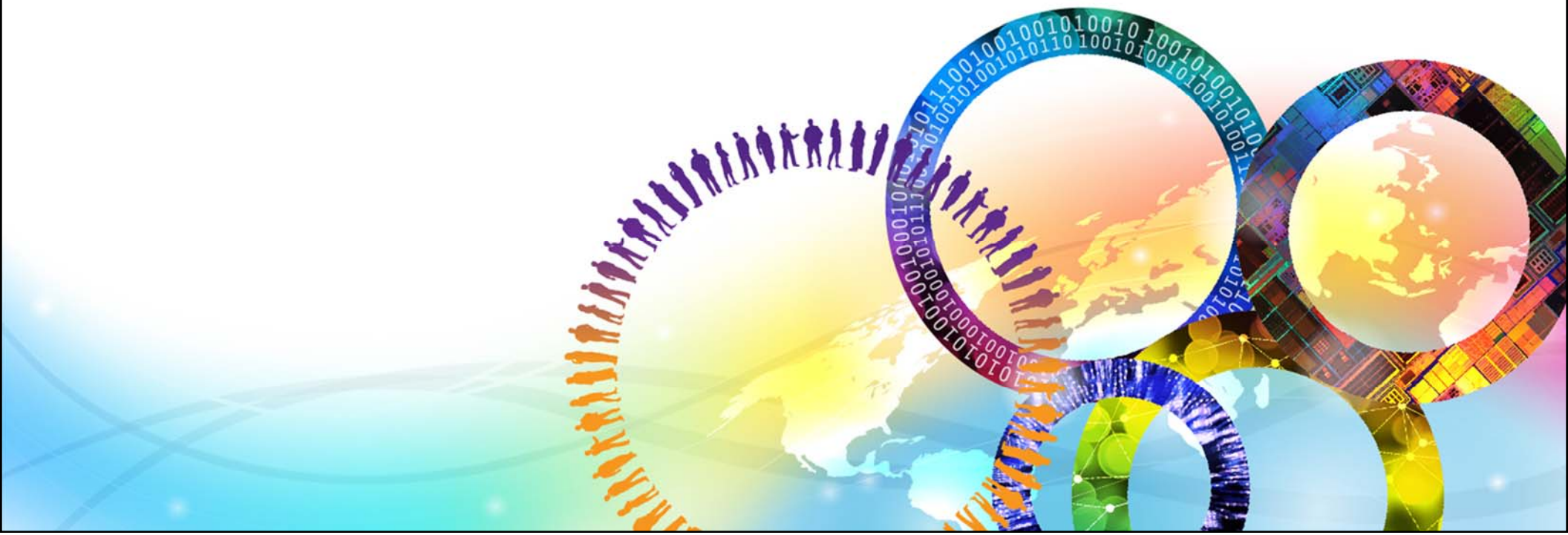
If you liked the presentation,
these colleagues deserve recognition

If you DID NOT LIKE **the presentation,**
these colleagues *deserve all the blame !!*

Sunburst Design

snug
Synopsys Users Group
Austin

Thank You



UVM Analysis Port Functionality and Using Transaction Copy Commands

Clifford E. Cummings
Sunburst Design, Inc.

Heath Chambers
HMC Design Verification, Inc.

World-class SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!

October 23, 2018
Austin