2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

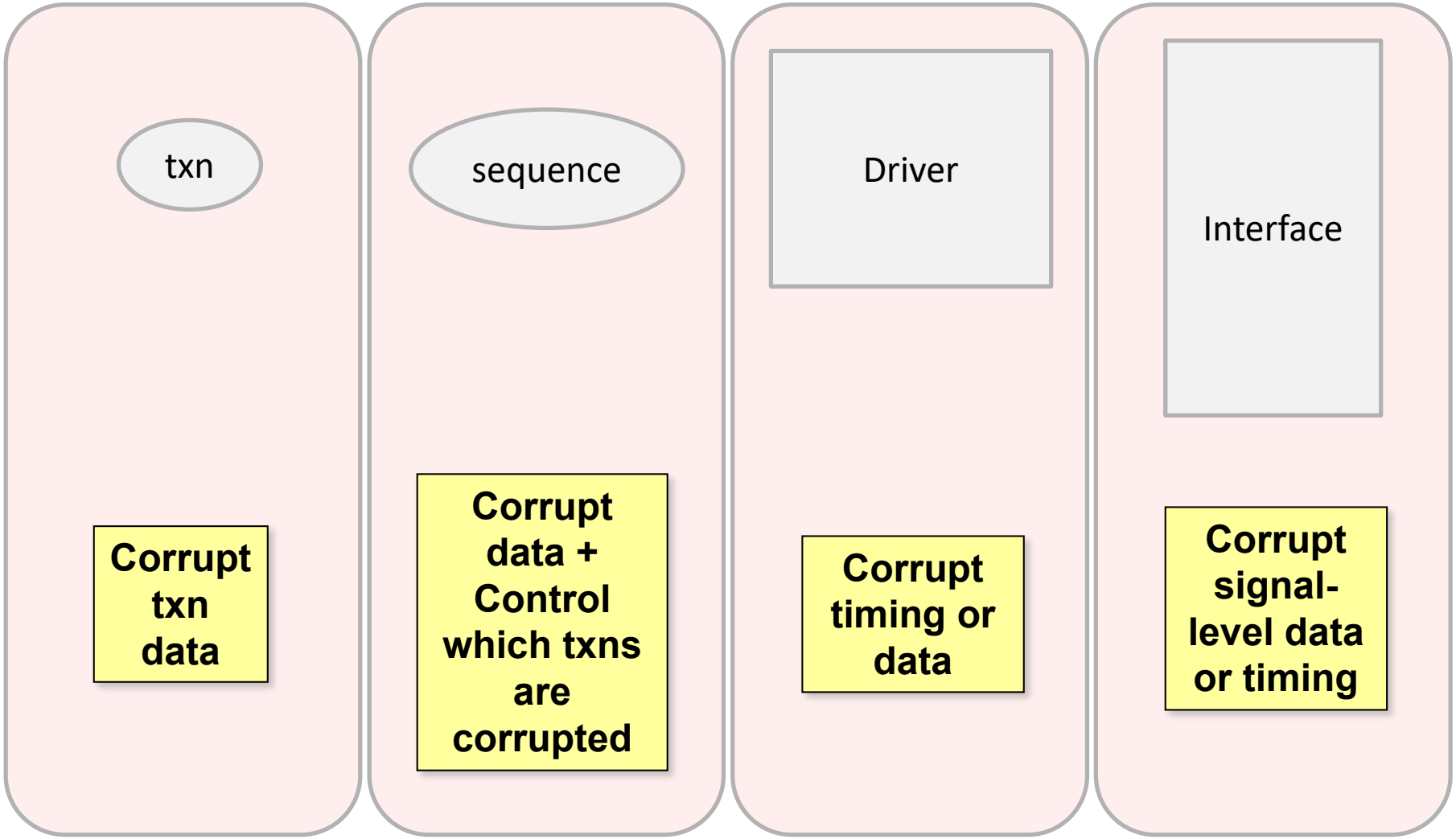
Error Injection: When Good Input Goes Bad

Kurt Schwartz, Aletheia Design Services LLC
Tim Corcoran, Willamette HDL, Inc.

Error Injection

- Deliberately generating stimulus that does not meet the requirements for the DUT
- Used to evaluate DUT behavior when input is unexpected or incorrect
  - Graceful handling of bad input is an indicator of design quality
  - Spec should specify behavior
    - Error registers
    - Interrupts
    - Write status record to memory

Error Injection in Stimulus



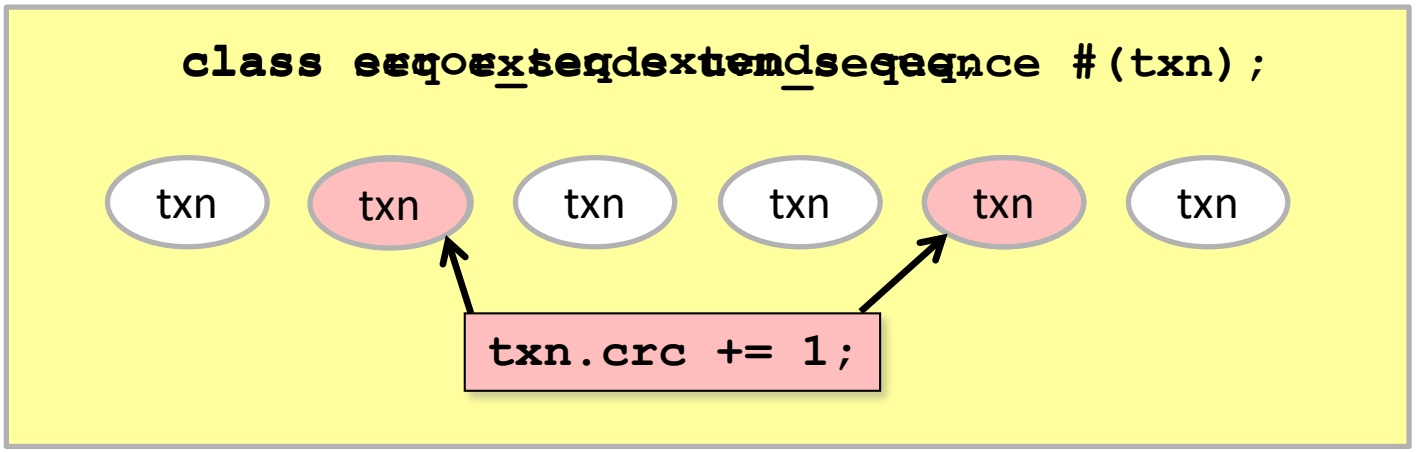
Error Injection in Transaction

```
class xtn extends uvm_sequence_item;

  virtual function void calc_crc();
     super.calc_crc();
     crc = crc + 1;
  endfunction
```

Test can use factory override to replace normal txn in a sequence

Errors are limited in scope to transaction data

Error Injection in Sequences



Test can use factory override to replace sequence

Sequence can have more control over error injection

Can't control any stimulus created in the driver

Scalability

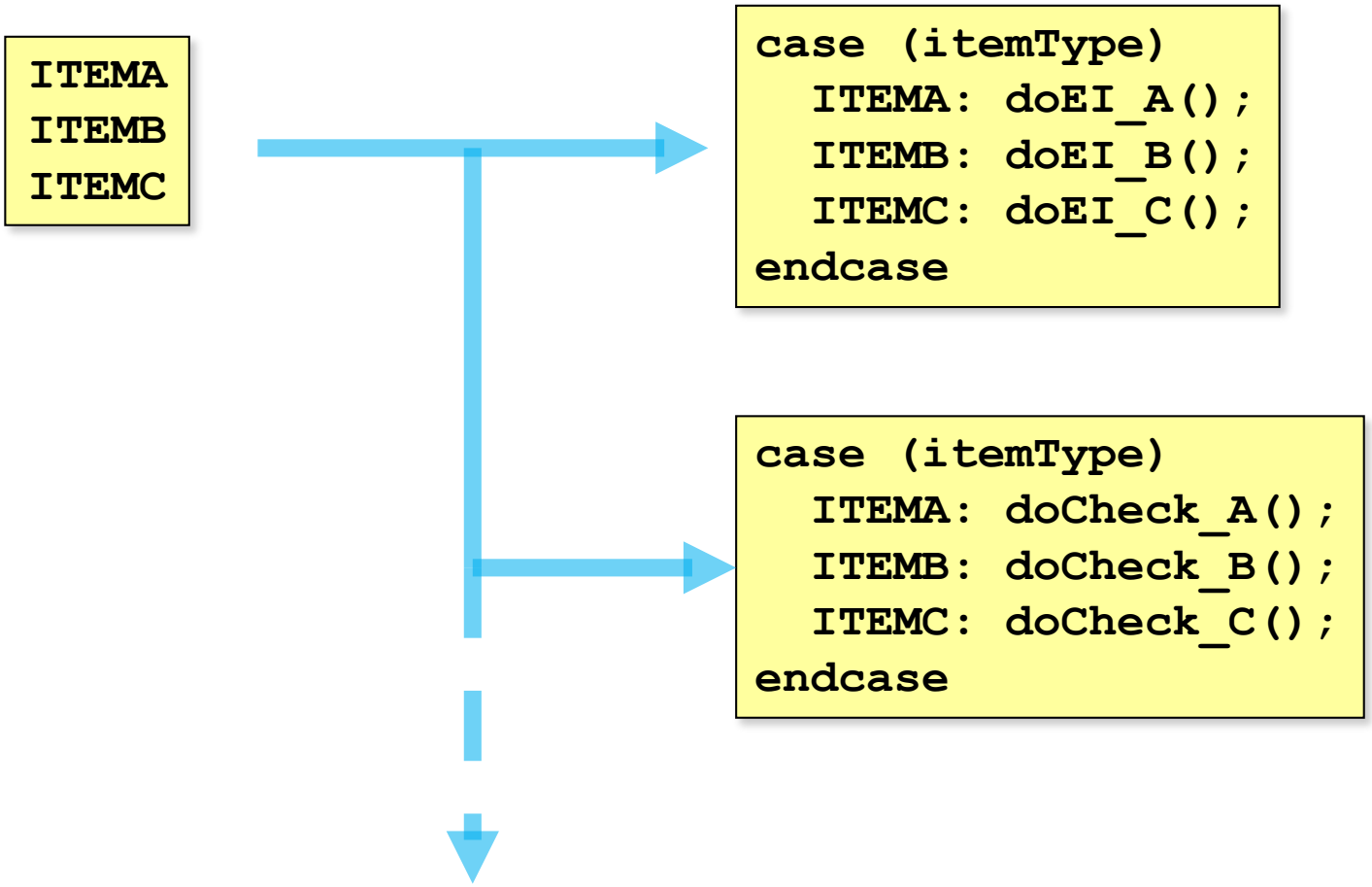
Error Injection in Driver

```
class driver extends uvm_driver #(txn);


    task run_phase(uvm_phase phase);
       . . .
       case (txn.errInj)
         ERR_NONE: drive_normal();
         ERR_CRC:  txn.crc += 1;
       . . .
    endtask
```

Most flexibility with error injection – best place to perform injection

Scalability: New error types require code changes in multiple places

Coding Pattern with Enums



Error Injection Objects

- Error Objects rather than enumerated type
- Derived from a base class
  - Base class defines error API

```
class error_injector_base extends uvm_object;
  . . .
  virtual function void inject(ref blockstream blocks
                               xlgmii_driver driver);
  endfunction
endclass
```

Pass in data to be driven

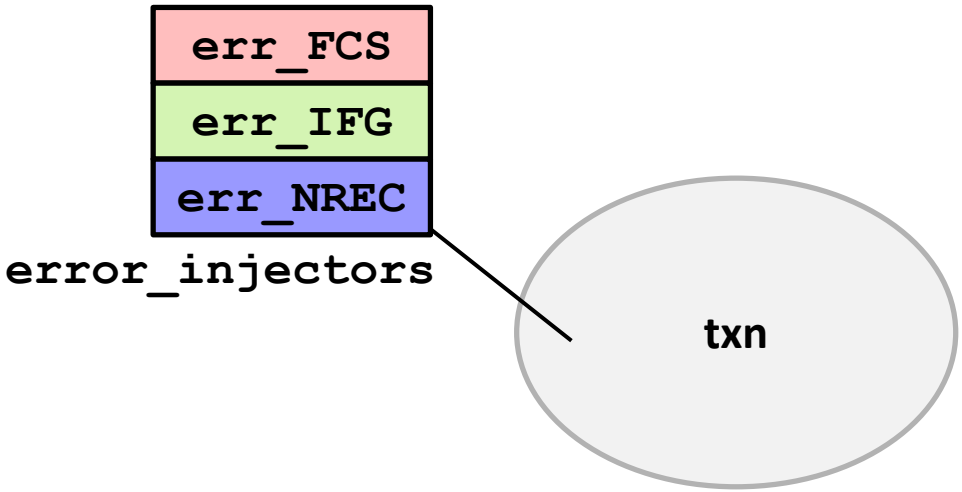
Driver can provide utility API, access to interface

Error Injection Objects

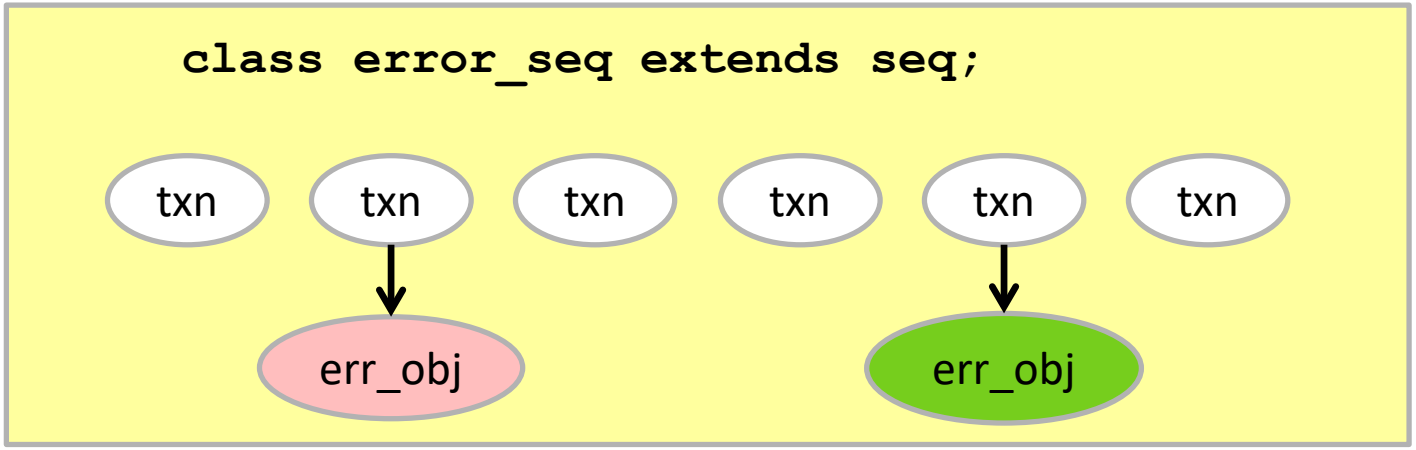
```
class ei_FCS extends error_injector_base;
  function void inject(ref blockStream    blocks,
                       xlgmii_driver  driver);
    sif_eop_txn fcsBlock;

 // Find the EOP record by its TXC value
    foreach (blocks[i]) begin
      if (blocks[i].control == 8'hF0) begin
        $cast(fcsBlock, blocks[i]);   break;
      end
    end

 // Corrupt the FCS
    fcsBlock.fcs[7:0] = ~fcsBlock.fcs[7:0];
  endfunction
endclass
```

Error Objects in Txn

```
class txn extends uvm_sequence_item;

  . . .

  error_injector_base error_injectors[$];

  . . .
endclass
```



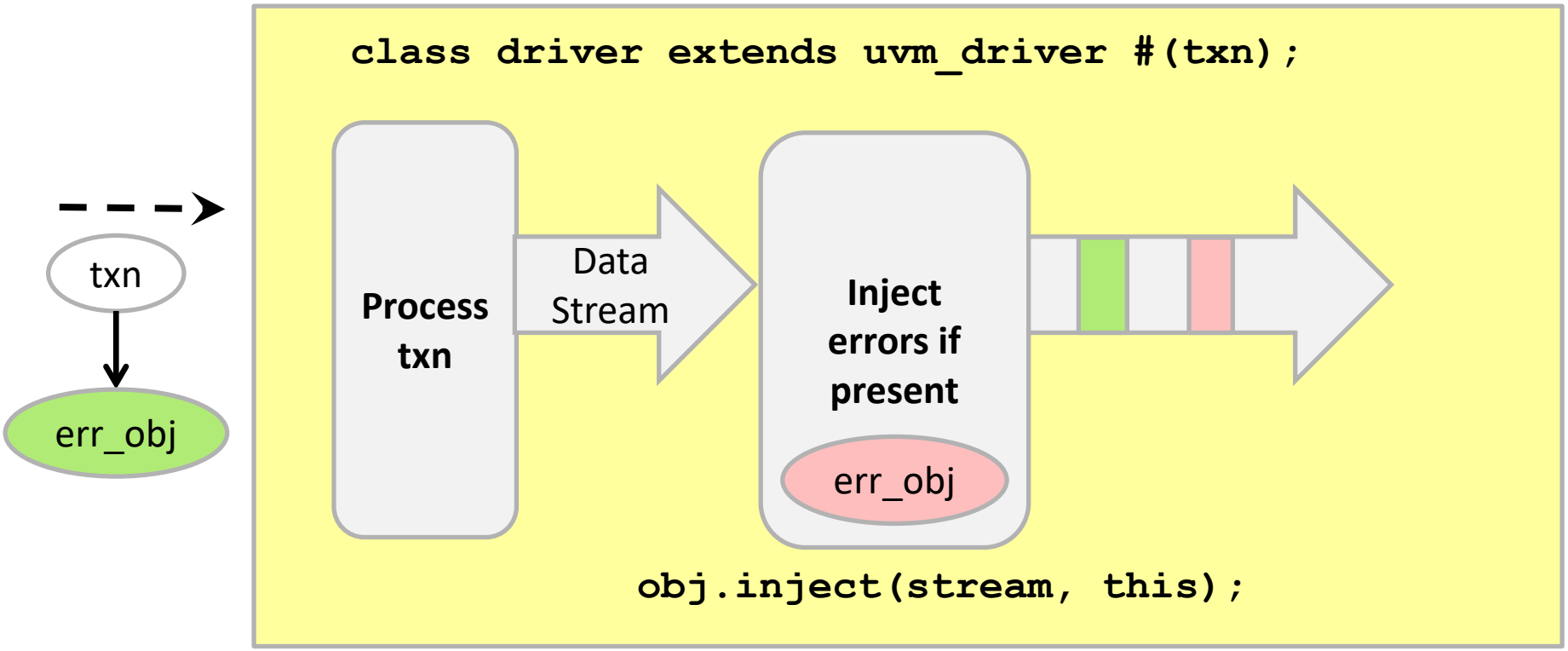
Error Selection in Sequences



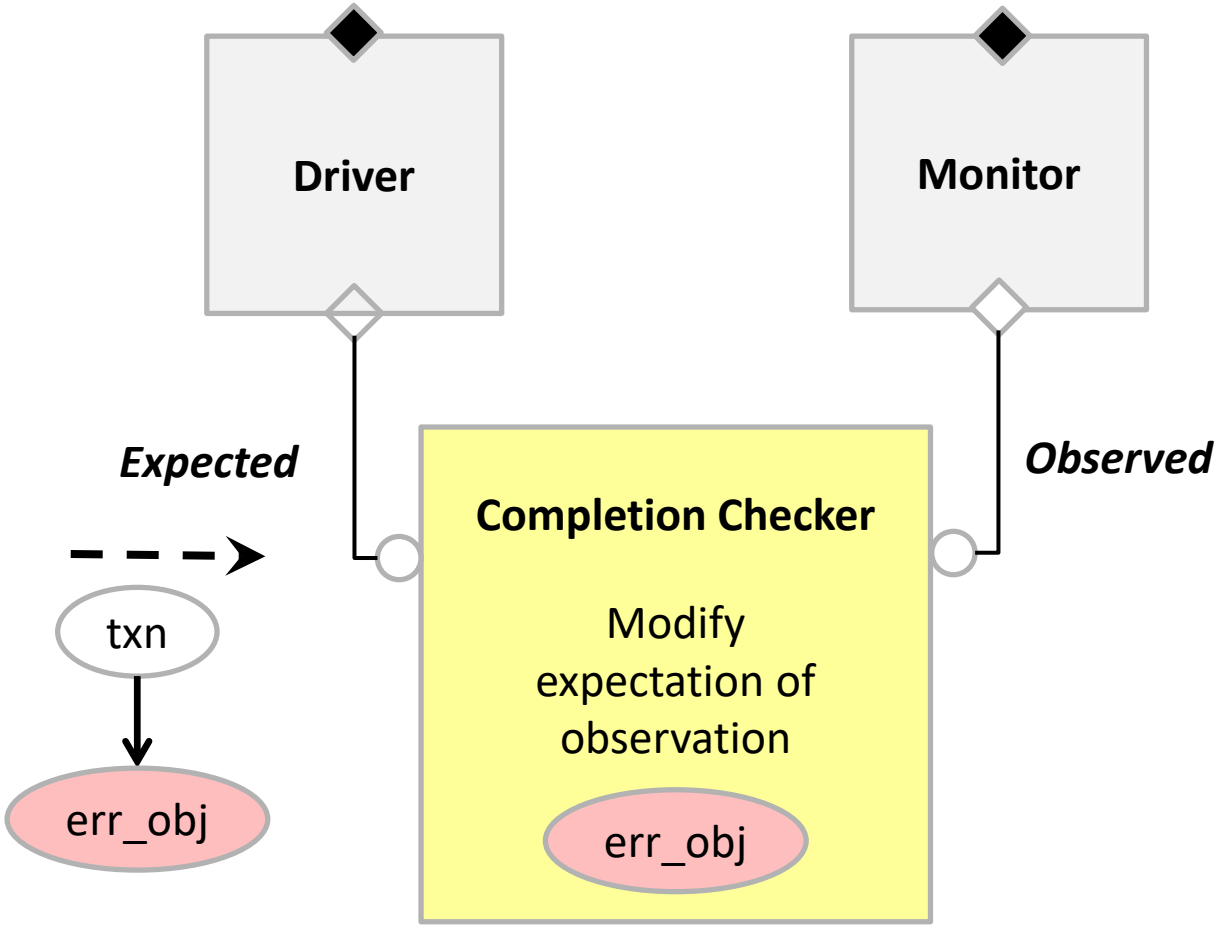
Test sequence can create and assign error injection objects

```
err_FCS eifcs;
err_IFG eiifg;

eifcs = err_FCS::type_id::create("eifcs");
eiifg = err_IFG::type_id::create("eiifg");

txn1.error_injectors.push_back(eifcs);
txn4.error_injectors.push_back(eiifg);
```

Error Injection in Driver



Error Injection in Scoreboard




```
obj.prepare_for_completion_error(this);
```

Scoreboard Error Injection

```
class ei_FCS extends error_injector_base;
       // Same as above...

   function void prepare_for_completion_error(
                        completion_checker complChecker);

     // This injection will cause STATUSREG.ERR = 1
      complChecker.expected_error_state = 1;
      complChecker.expected_error_code = 8'h05; // FCS
   endfunction

endclass
```

Summary

- More flexible
  - Injection in driver provides access to driver and interface API
    - Timing as well as data errors can be injected
    - Errors can be applied over multi-transaction context
- Object-based vs. enum-base provides better reuse
  - All functionality is encapsulated
  - Same technique can apply to scoreboards