

Modeling Usable & Reusable Transactors in SystemVerilog

Janick Bergeron, Scientist
Verification Group, Synopsys Inc
janick@synopsys.com



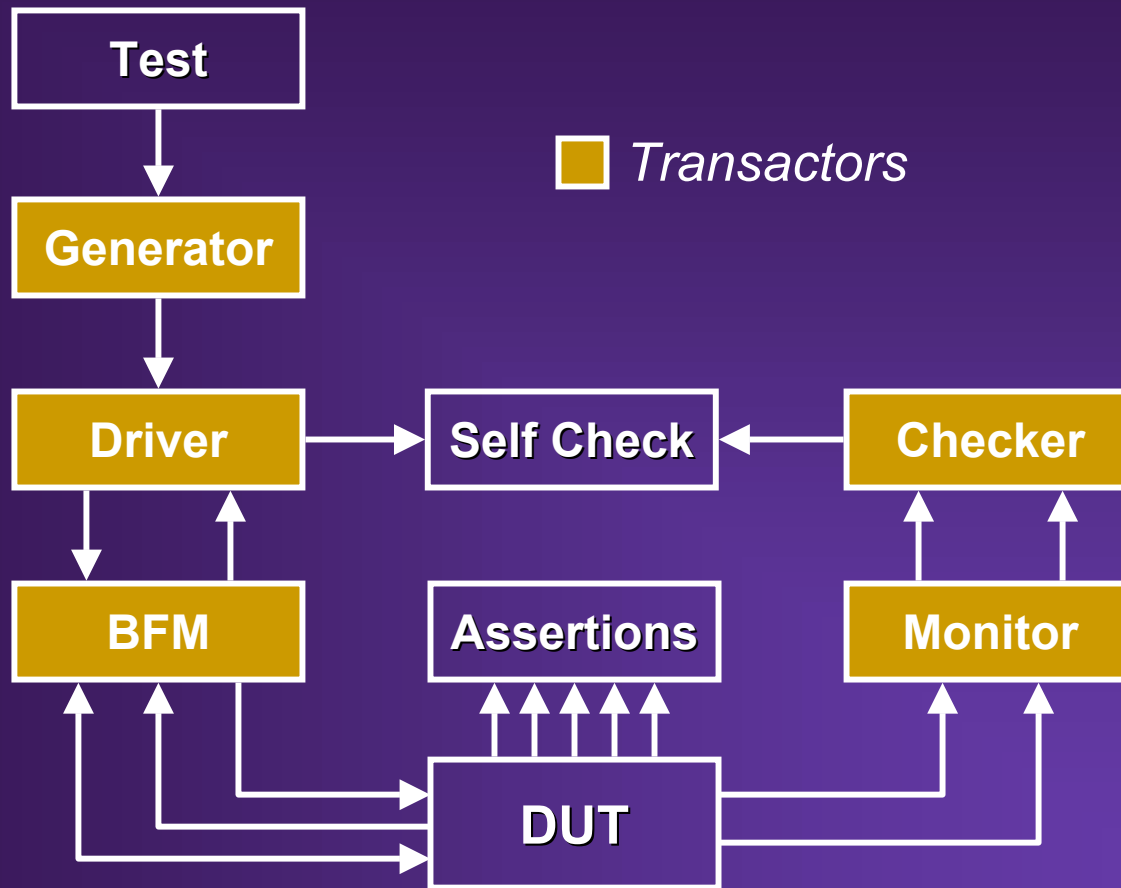
Transactors

Definition

- Building blocks of verification environments
 - Bus-functional models
 - Monitors
 - Checkers
 - Generators
- Few instances
 - Created at the start of simulation
 - Remain in existence for duration
- Data and transactions flow through them
 - Generation
 - Observation
 - Scheduling
 - Transformation
 - Processing

Verification Environment

Transactors



Transactor

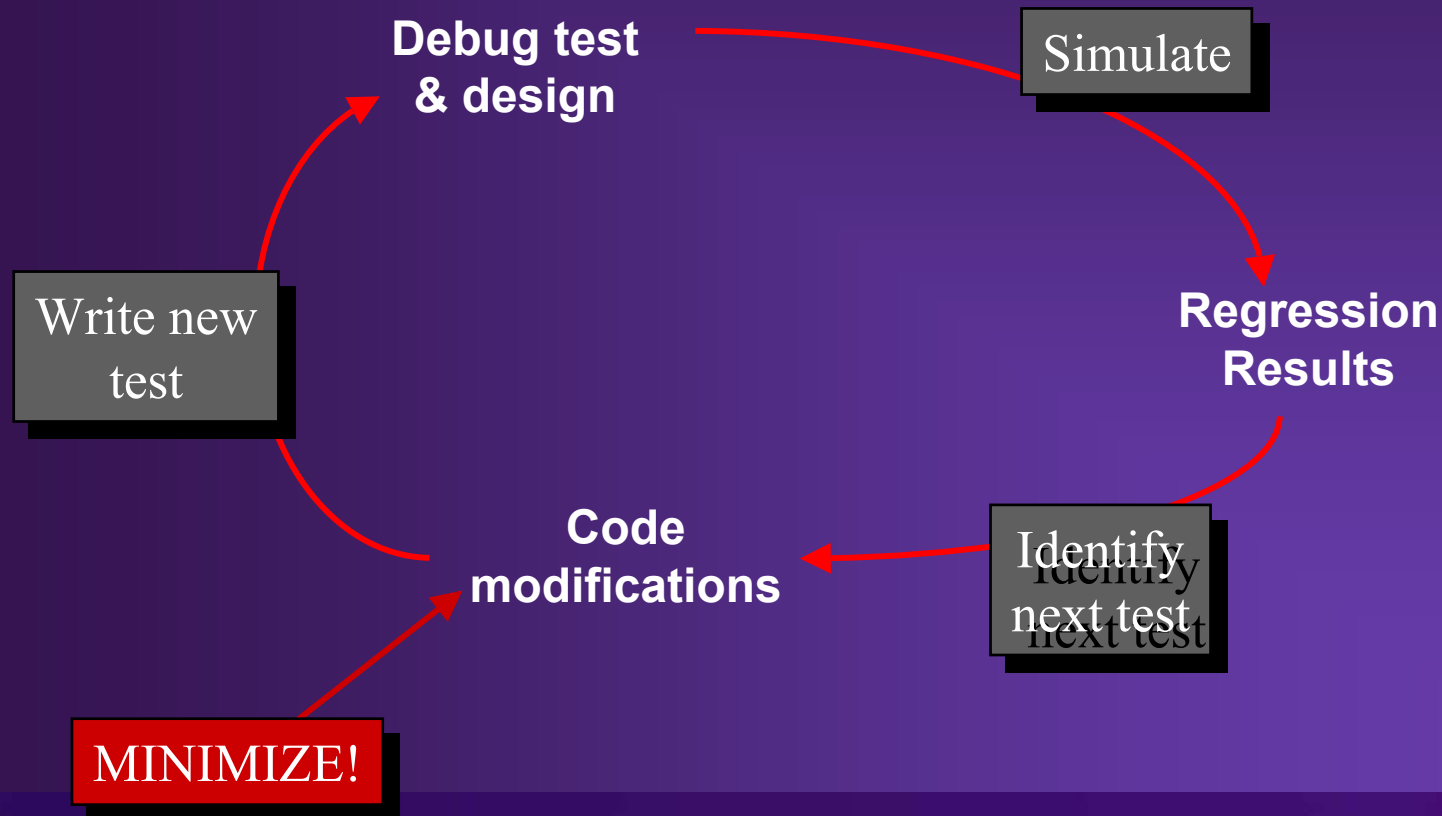
Categories

- Active transactors
 - Initiate transactions
 - Supply data to other side
 - Example: AHB master, Ethernet Tx
- Reactive transactors
 - Transaction initiated by other side
 - React by supplying requested data
 - Example: AHB slave
- Passive transactors
 - Transaction initiated by other side
 - Collect transaction data from other side
 - No interaction with monitored interface
 - Example: AHB bus monitor

Verification Process

Why Should You Care About Transactors?

- Verification >60% of design effort
- Transactors enable abstraction & automation



Transactors

Implementation

- Simple to write transactors that
 - Always do the right thing
 - Operate as fast as possible
- How can an environment
 - Integrate a scoreboard?
 - Add functional coverage?
- How can a test
 - Introduce delays?
 - Respond with "retry", "abort" or not respond?
 - Inject errors?

Without rewriting/modifying
original transactor

Traditional Transactors

Procedural Interface

- Complex control interface

- Configuration commands
- Transaction commands
- Exception commands

Different concerns
Same mechanism

- Creates new programming language

Rich set of commands

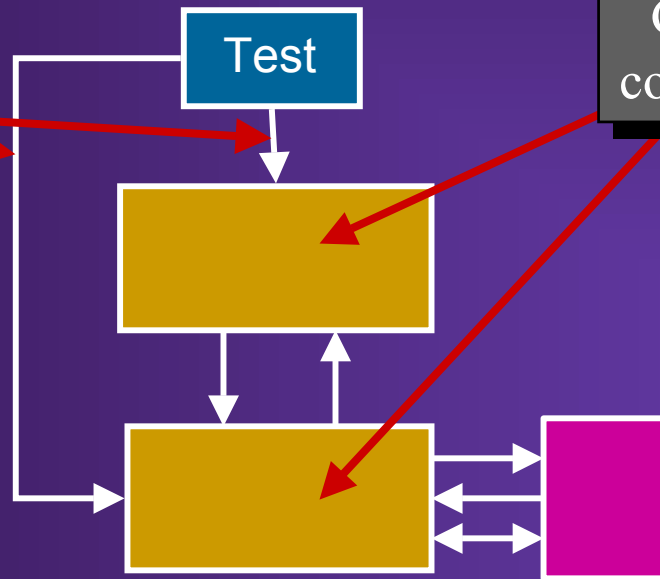
Test

Grow in complexity

Desired function must exist in a command

Always modifying transactor

Become test-specific



Usable & Reusable Transactors

- Separate
 - Configuration
 - Transactions
 - Exceptions
- Minimize code required for additional tests
- Support
 - Random configuration
 - Random stimulus
 - Self-checking operations
 - Functional coverage
 - User-defined exceptions
 - Block-level verification
 - System-level verification

Requires HVL
features


Object-Oriented Transactors

Configuration Interface

- Implement using *class*
- Configure using *configuration class*
 - Can be randomized
 - Pass via *constructor* and optional *reconfigure()* method

```
class mii_cfg;  
    rand bit is_100Mb;  
endclass  
  
class mii;  
    protected mii_cfg cfg;  
    ...  
    function new(mii_cfg cfg, ...);  
        this.cfg = cfg;  
    endfunction;  
    function void reconfigure(mii_cfg cfg);  
        ...  
    endfunction;  
endclass
```

May have to
reset transactor



Object-Oriented Transactors

Transaction Interface

- Data-flow based transaction interface
 - Transaction descriptors
 - See "*Modeling Data and Transactions*", DesignCon '05

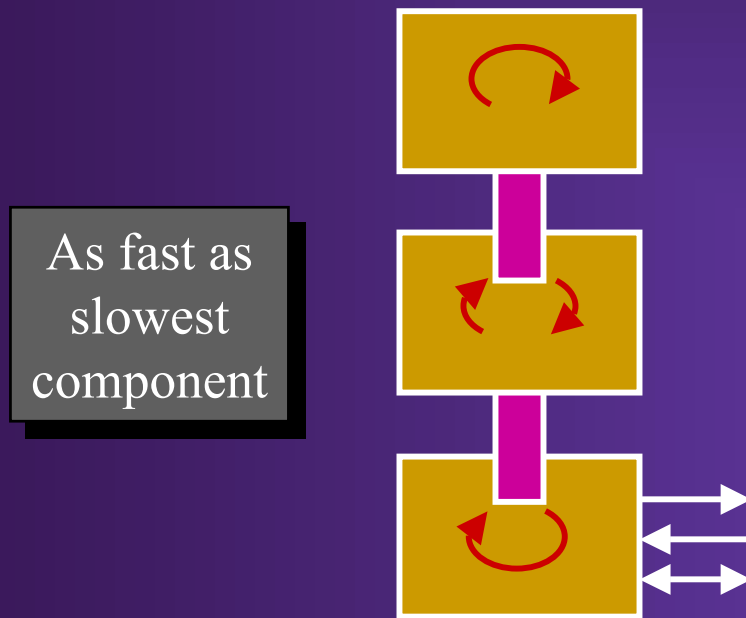
```
class mii;
    eth_frame_channel in_chan;
    eth_frame_channel out_chan;
    ...
    function new(..., eth_frame_chan in_chan = null,
                eth_frame_chan out_chan = null, ...);
        if (in_chan == null) in_chan = new;
        this.in_chan = in_chan;

        if (out_chan == null) out_chan = new;
        this.out_chan = out_chan;
        ...
    endfunction;
endclass
```

Object-Oriented Transactors

Transaction Interface

- Flow of data between producer and consumer automatically regulated



Object-Oriented Transactors

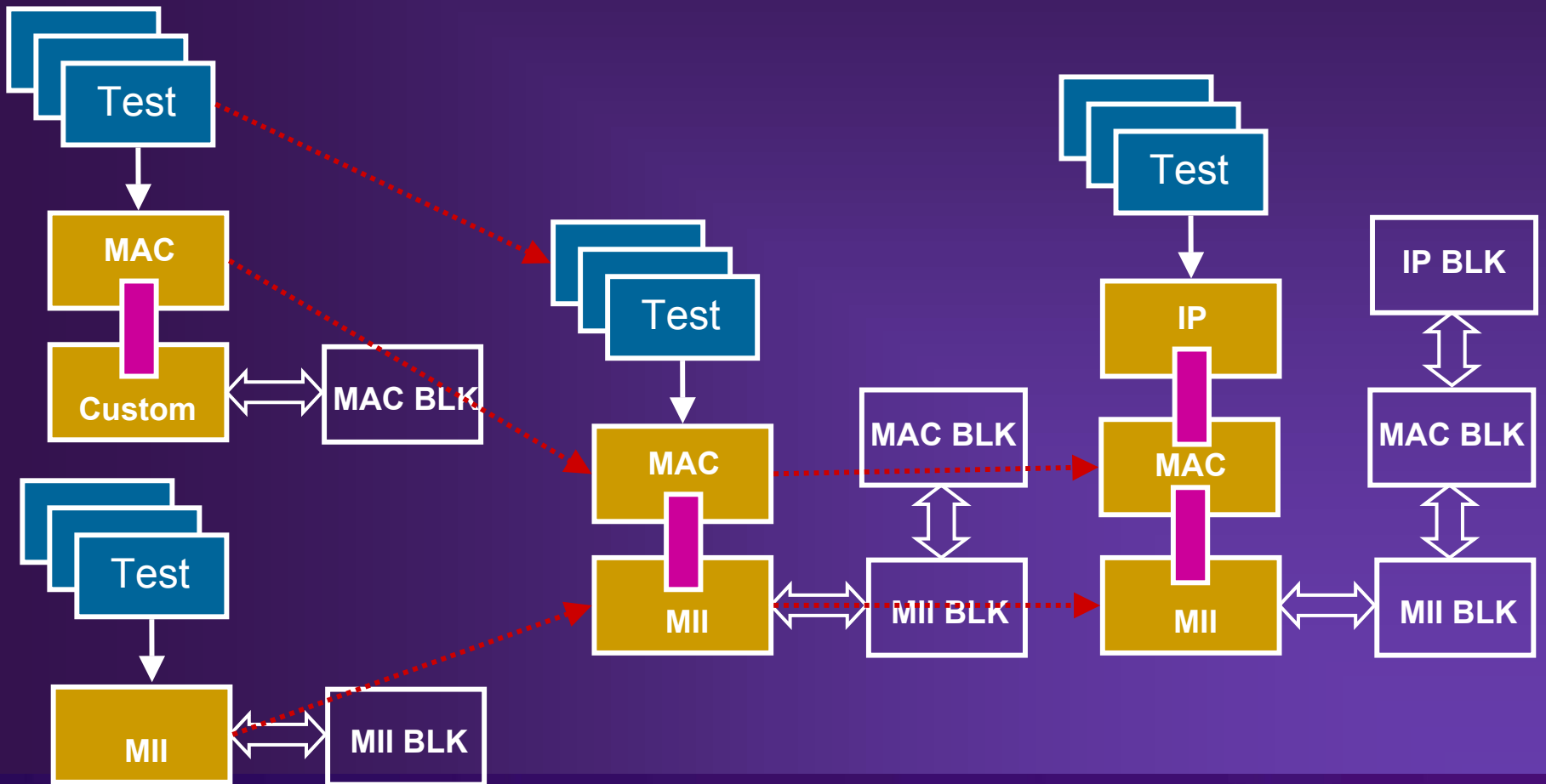
Transaction Interface

- Can implement various transaction completion models
 - Blocking
 - Nonblocking
 - Out-of-order
 - Request / Response

Object-Oriented Transactors

Transaction Interface

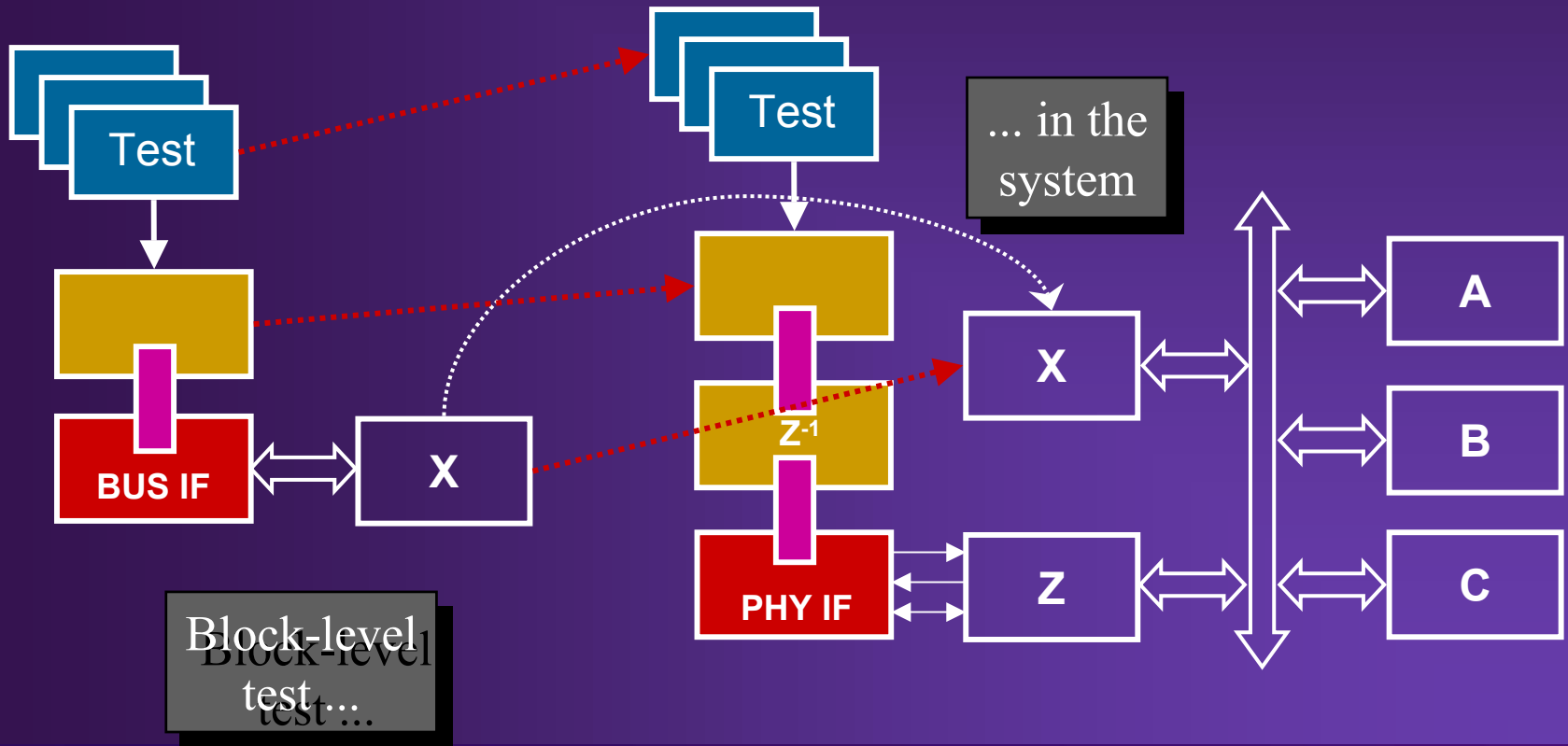
- Key to transactor reuse across environments



Object-Oriented Transactors

Transaction Interface

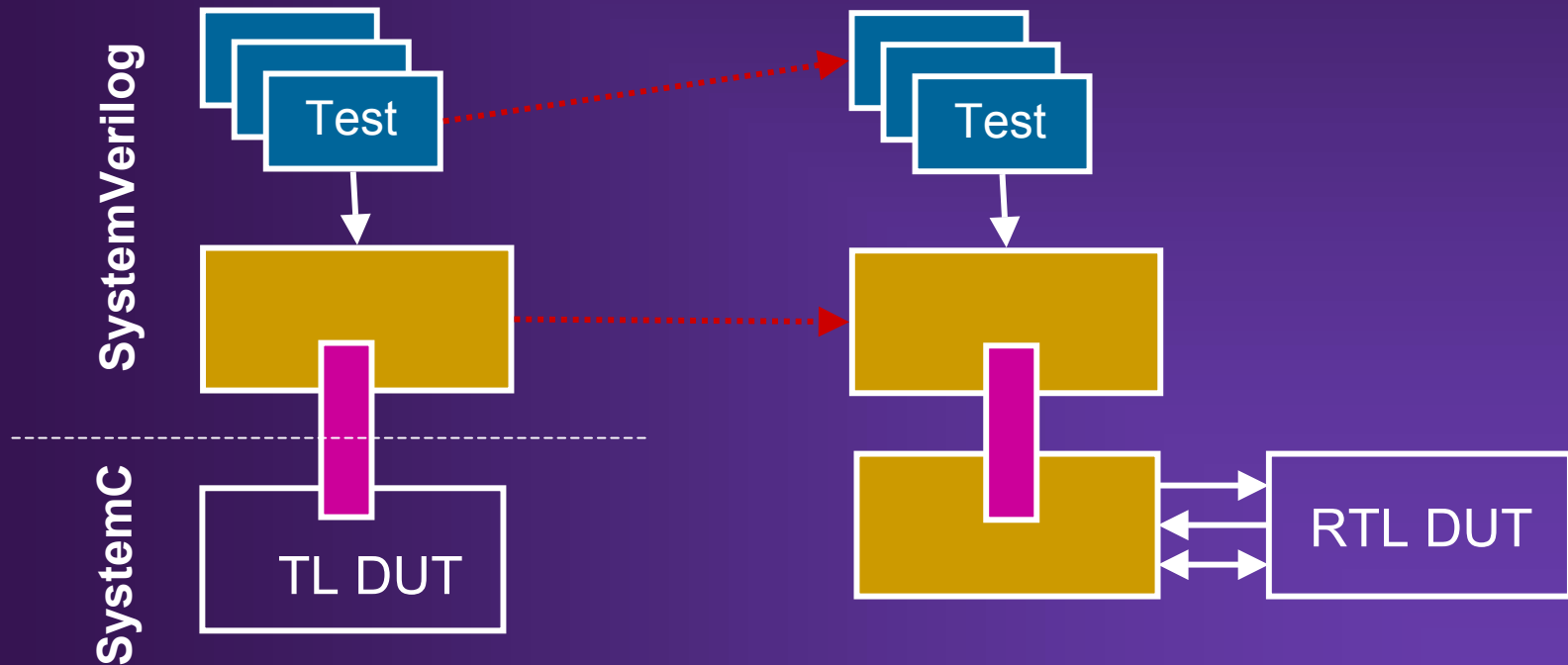
- Allows port of block level tests to chip level



Object-Oriented Transactors

Transaction Interface

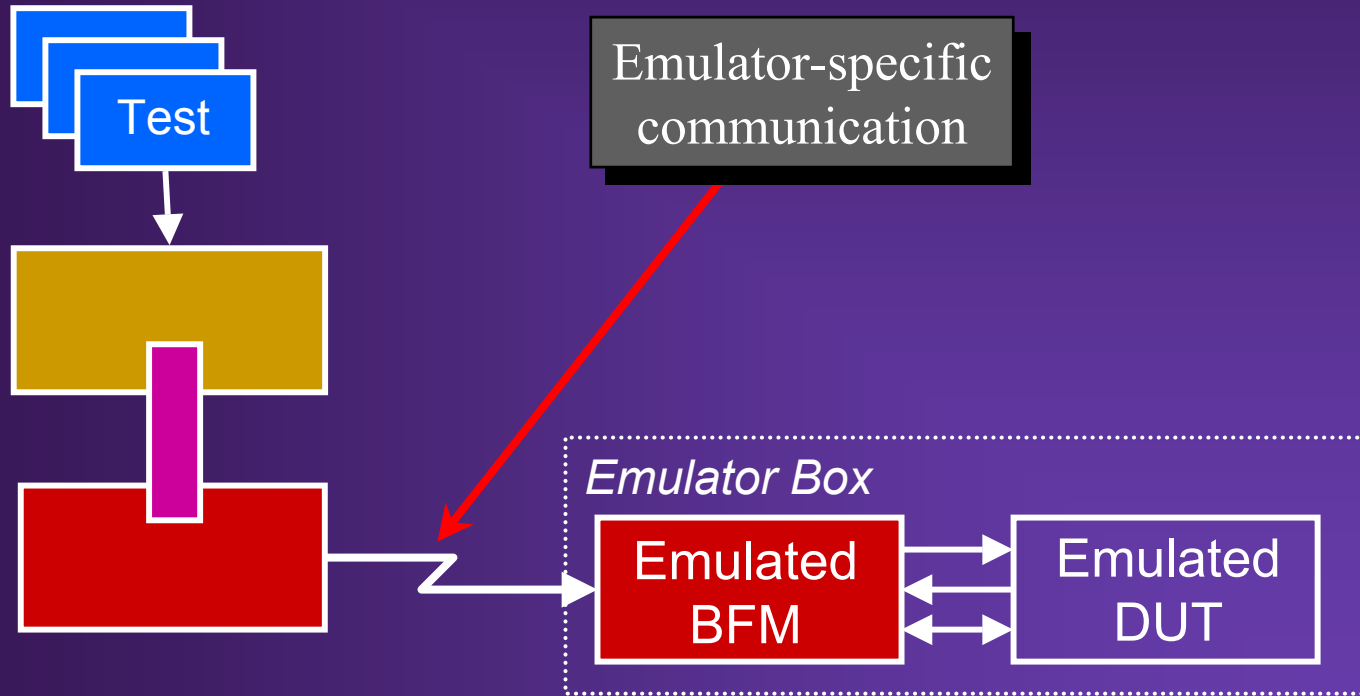
- Components unaware of other endpoint



Object-Oriented Transactors

Transaction Interface

- Interface between testbench and emulator



Object-Oriented Transactors

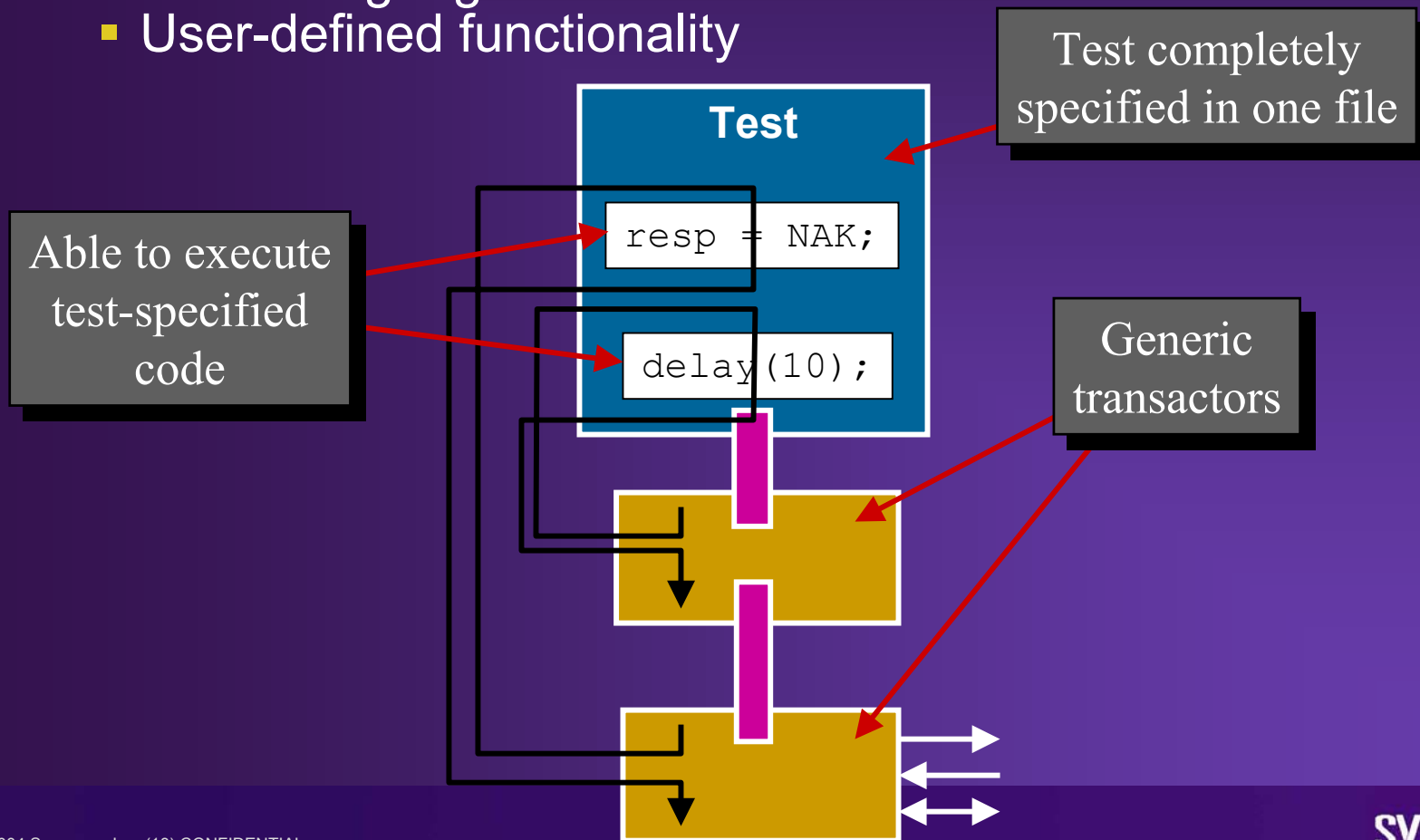
Exception Interface

- Reusable transactors need mechanism to provide services to verification environment
 - Introducing delays
 - Synchronizing different transactors
 - Feedback
 - Recording input into scoreboard
 - Comparing output with expected values
 - Sampling data for functional coverage
 - Modifying transactions to inject errors
 - Deviate from default behavior
- Flexible enough to satisfy unpredictable needs of verification environments and testcases
 - Without modifying transactor

Object-Oriented Transactors

Exception Interface

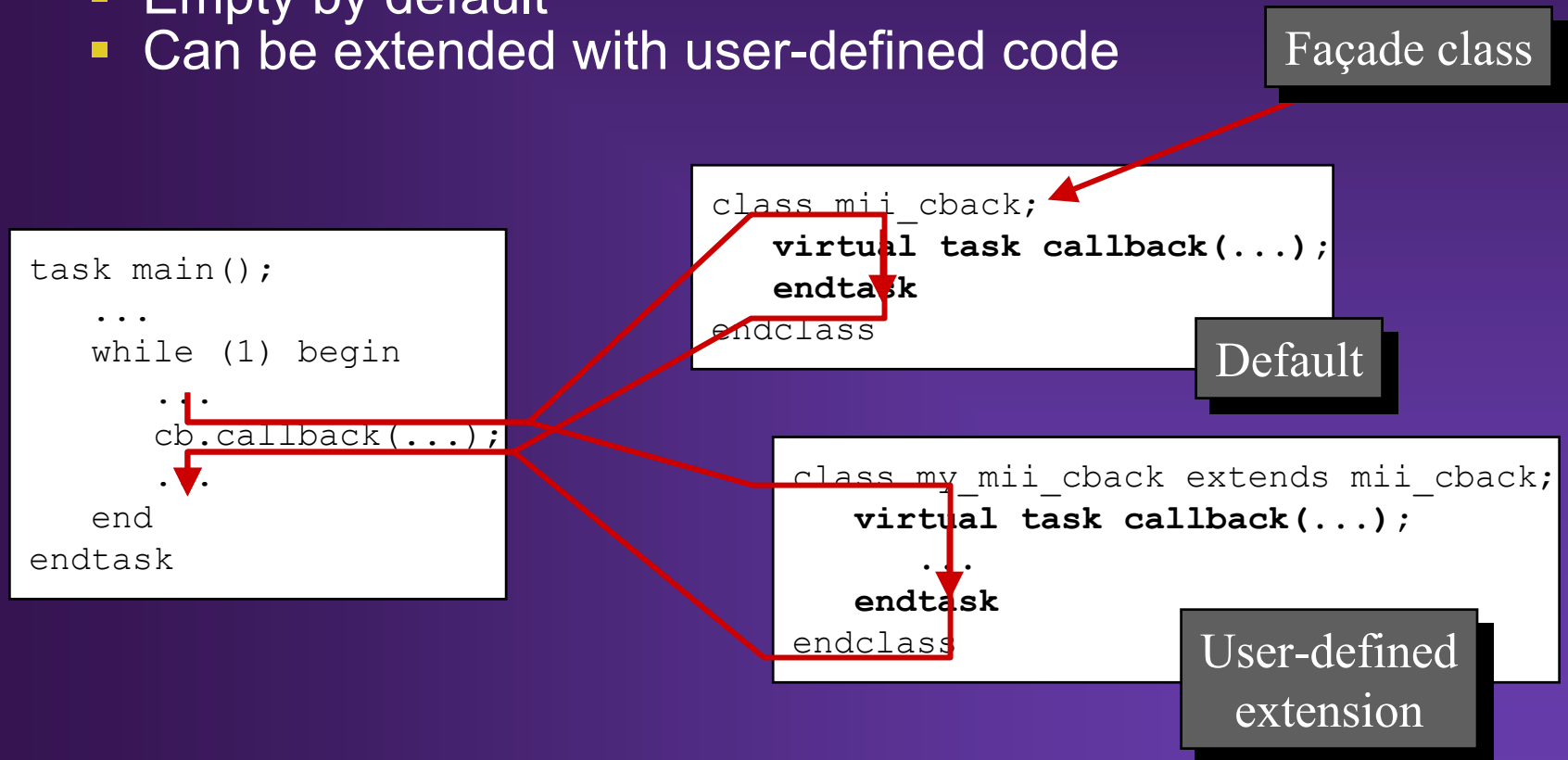
- Callbacks used to implement exceptions
 - Same language
 - User-defined functionality



Reusable Transactors

Callback Methods

- Callback methods are *virtual* methods in *façade* class invoked by the transactor itself
 - Empty by default
 - Can be extended with user-defined code



Reusable Transactors

Callback Methods

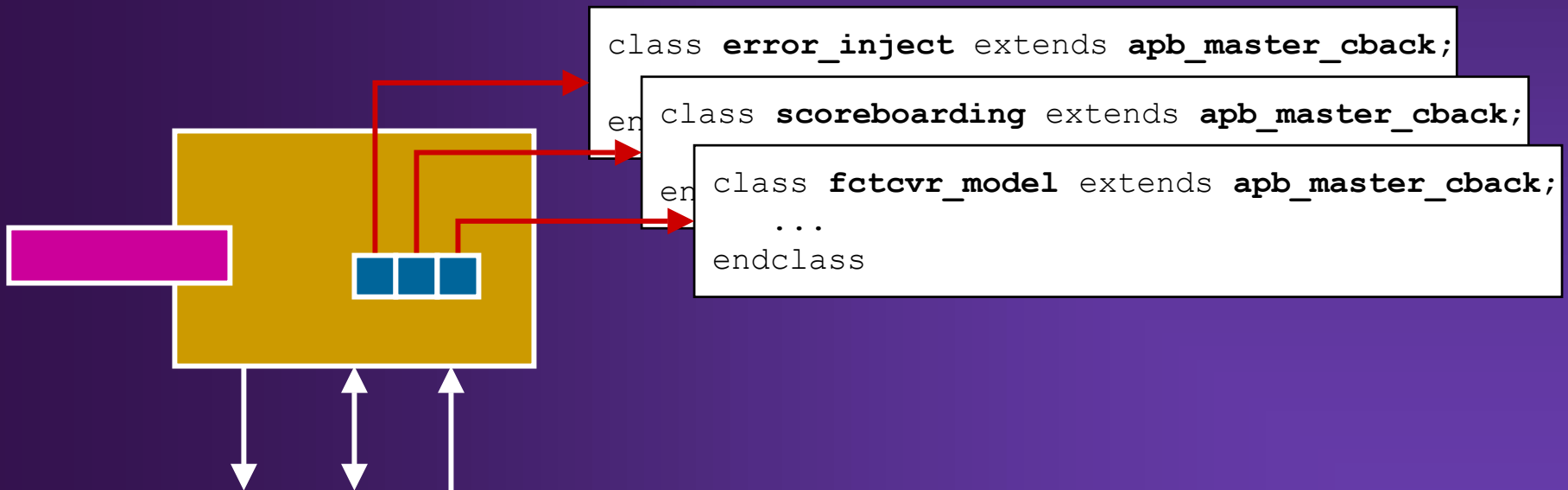
- Provide rich set of callback methods
 - After a new transaction is about to be started
 - Allow delay insertion
 - Allow modifying or dropping the transaction
 - Before a major decision is about to be acted upon
 - Allow the default decision to be changed
 - Before sub-transactions or data is transmitted
 - Allow delay insertion
 - Allow modifying or dropping
 - After sub-transactions or data has been received
 - Allow modifying or dropping
 - After a transaction has been completed
 - Allow modifying or prevent forwarding to higher layers
 - In the body of every loop
 - Supply loop index via argument
 - Allow modifying the subject of the iteration

If protocol allows it

Reusable Transactors

Callbacks Methods

- Transactors have list of registered callback extensions
 - Individual extensions for different purposes
 - No modifications of existing callback functionality



Extensible Transactors

User-Defined Transactions

- Use callback to interpret transaction descriptor
 - Can be overloaded to modify transaction execution
 - Can be overloaded to define new transactions

```
task main();
...
while (1) begin
    transaction tr;
    in_chan.get(tr);
    ..
    if (cb.exec(tr)) next;
    ..
end
endtask
```

```
class mii_cback;
    virtual function bit exec(transaction tr);
        exec = 0;
    endfunction
endclass
```

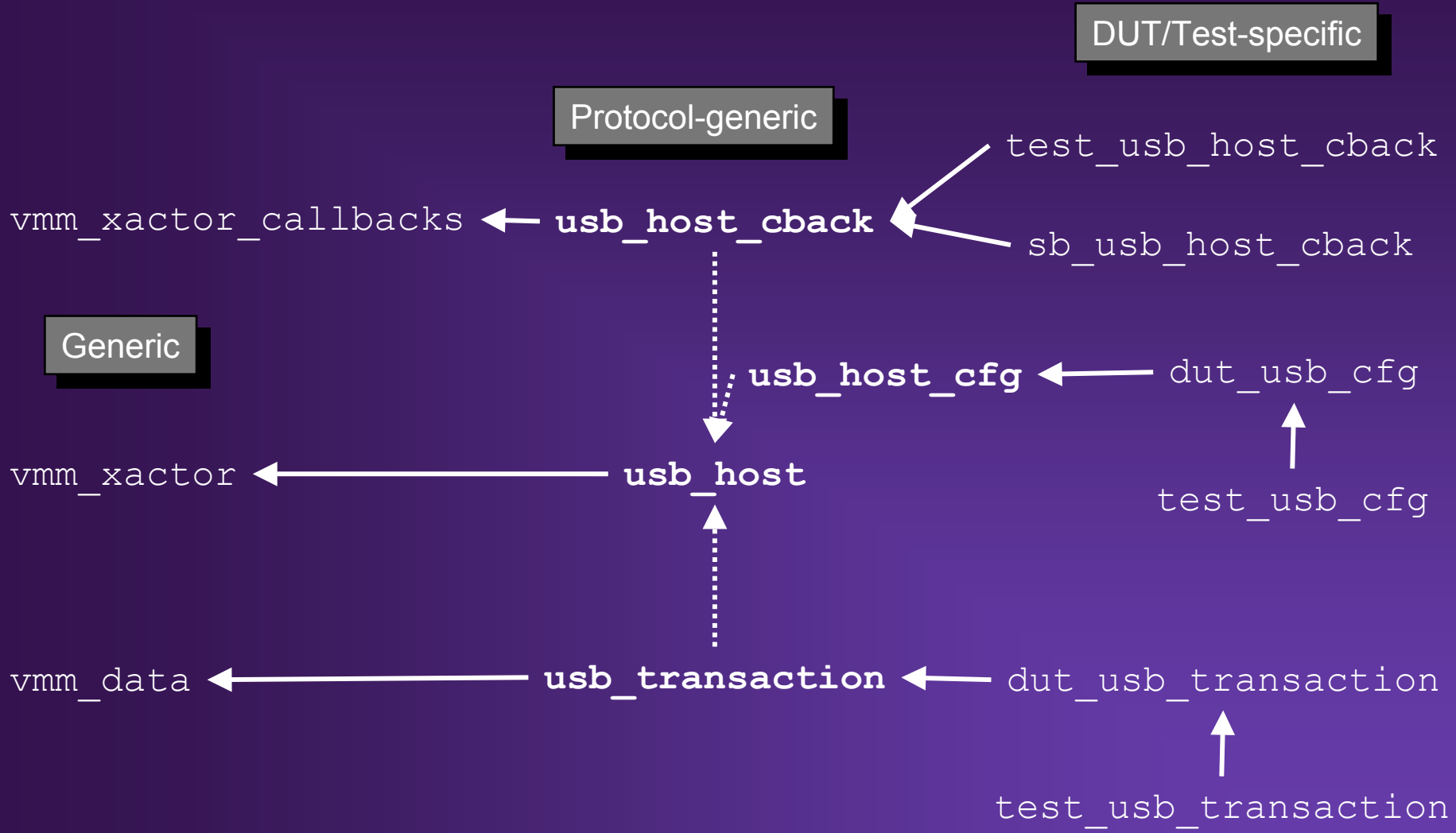
Default

```
class my mii_cback extends mii_cback;
    virtual function bit exec(transaction tr);
        exec = 0;
        if (tr.kind = MY_TRANSACTION) begin
            ..
            exec = 1;
        end
    endfunction
endclass
```

User-defined extension

Transactor Implementation

Class Hierarchy



Useable & Reusable Transactors

For more information

- SystemVerilog Verification Methodology Manual
 - Kluwer Academic Publishers, 2005
- "*Modeling Data and Transactions*"
 - DesignCon '05, Santa Clara, CA
- Reference Verification Methodology
 - Vera User's Guide, 6.2 and later

Modeling Usable & Reusable Transactors in SystemVerilog

Janick Bergeron, Scientist
Verification Group, Synopsys Inc
janick@synopsys.com

