



**Open Verification
Methodology**



**Universal Verification
Methodology**

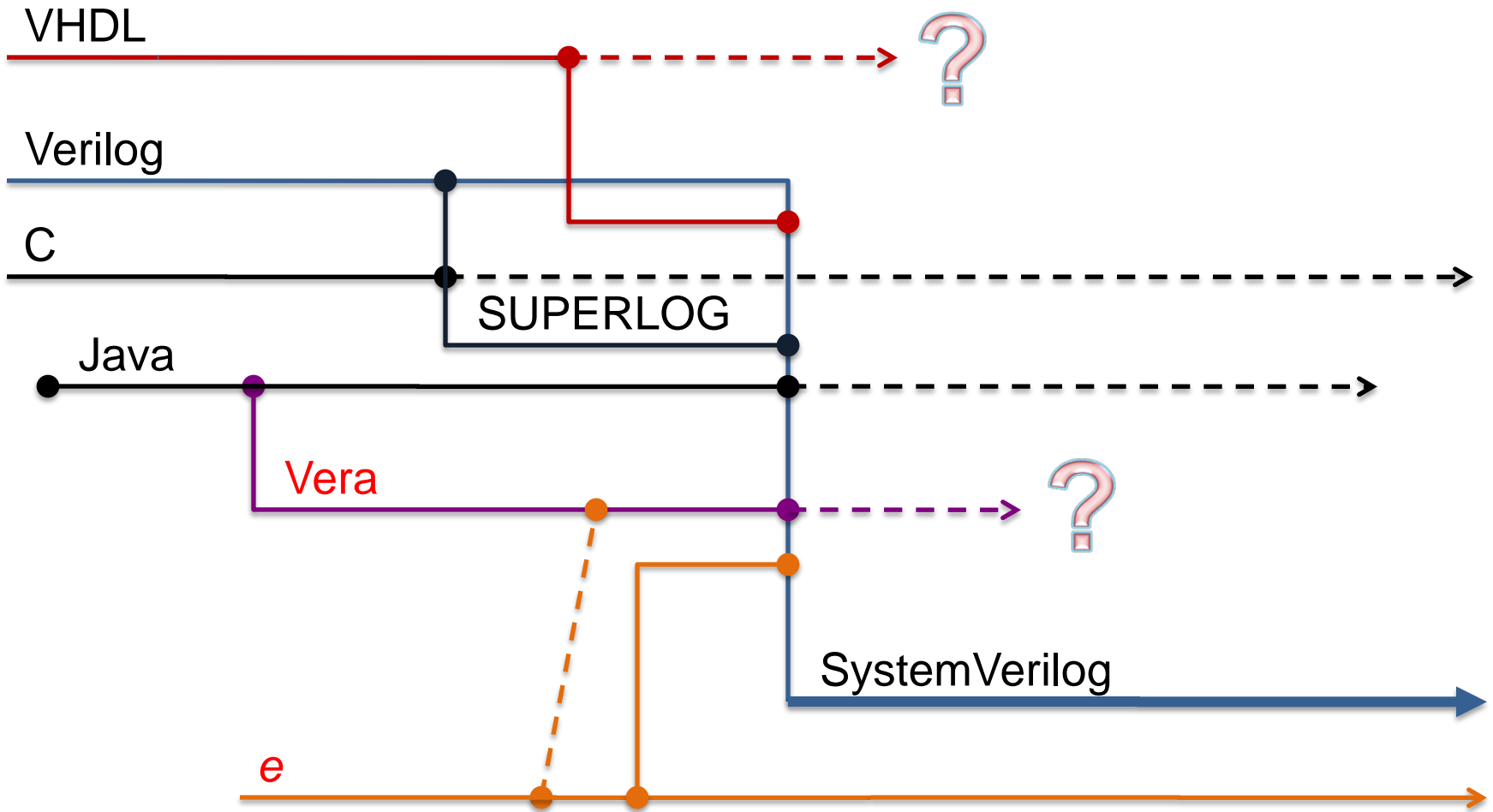


OVM/UVM Update

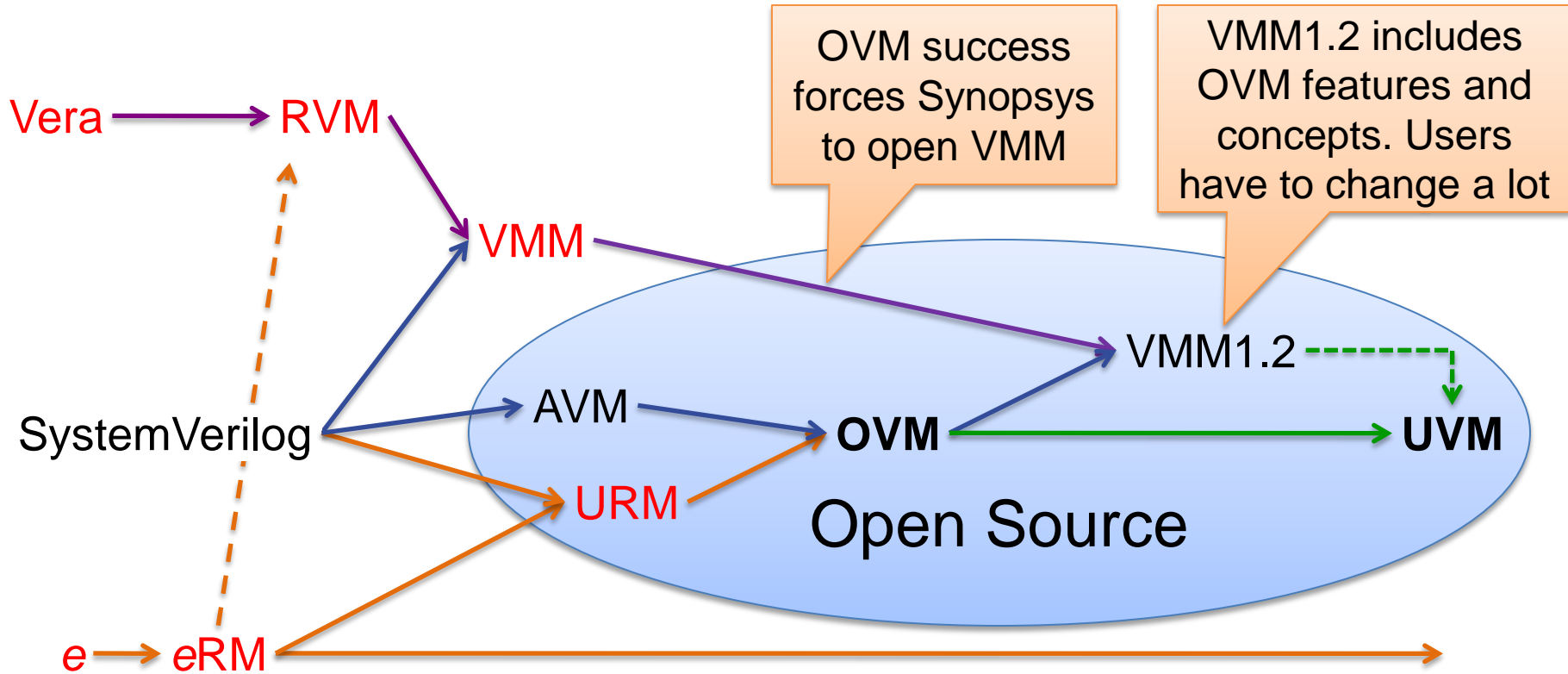
Tom Fitzpatrick
Verification Technologist
Mentor Graphics Corp.

Sharon Rosenberg
Solutions Architect
Cadence Design Systems

Language History



Methodology History



The Path to Accellera UVM: It All Started...

- **Accellera formed the VIP-TSC in May, 2008**
- **Short-term goal: Create an OVM-VMM interoperability solution**
- **Status**
 - Implementation complete: May'08
 - Document complete: July'08
 - Document approved: August'08
 - Kit released: December'08
- **Long-term goal: Create a single verification methodology that everyone will use**

VIP-TSC Organization & Process

■ **Voting Member Companies**

- Aldec, Cadence, Cisco Systems, Denali, Freescale, IBM, Intel, Mentor Graphics, Nokia, Paradigm Works, SpringSoft, ST Microelectronics, Sun, Synopsys, Texas Instruments, Verilab, Xilinx

■ **December 2009 Vote**

- Chose OVM as starting point
- Rename OVM to UVM
- OVM backward compatibility is a key goal
- Include other contributions as appropriate

■ **March 2010 Face-to-Face Meeting**

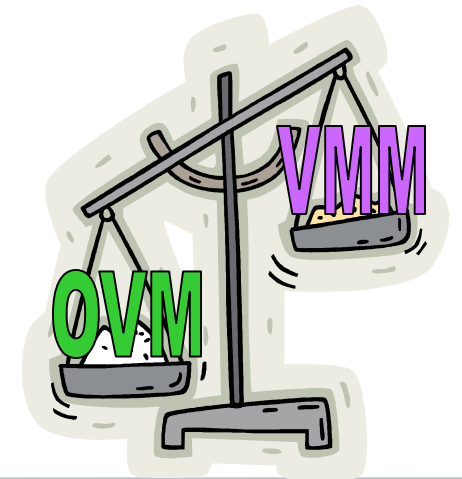
- Agreed to “Top 10” features for UVM1.0
- Register Package and Phasing are Top 2

UVM is OVM

- **UVM1.0 Early Adopter Kit released May 17, 2010**
- **Based on OVM2.1.1**
 - “ovm” -> “uvm”
 - “tlm” -> “uvm_tlm”
 - callback & objection enhancements
 - message catching
- **“Script compatible” with OVM2.1.1**
 - Trying UVM-EA is as easy as running a script



OVM Wins!



OVM/UVM: The Methodology Platform

■ Component/Environment/Test

- Structural hierarchy

■ Phasing

- Automatically ensure completion of necessary steps

■ TLM Communication

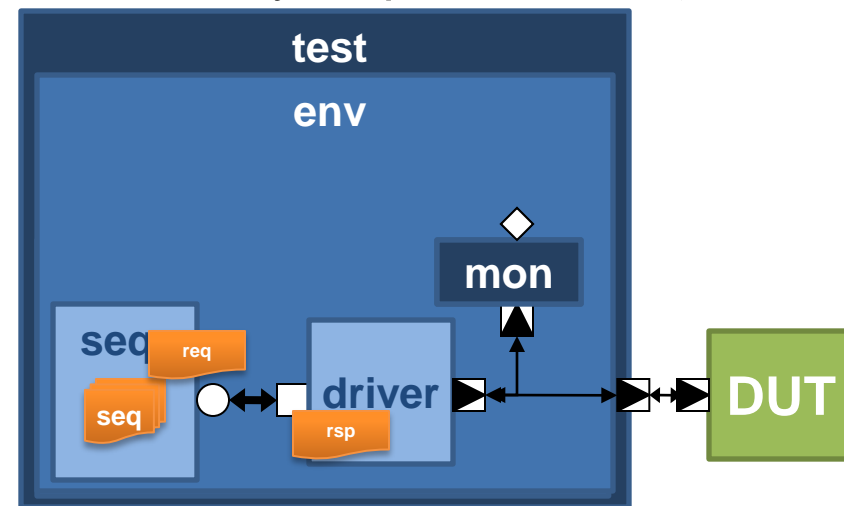
- Provides modularity and reuse

■ Factory

- Provides types/instance flexibility

■ Sequences

- Hierarchical stimulus
- Separate from design hierarchy



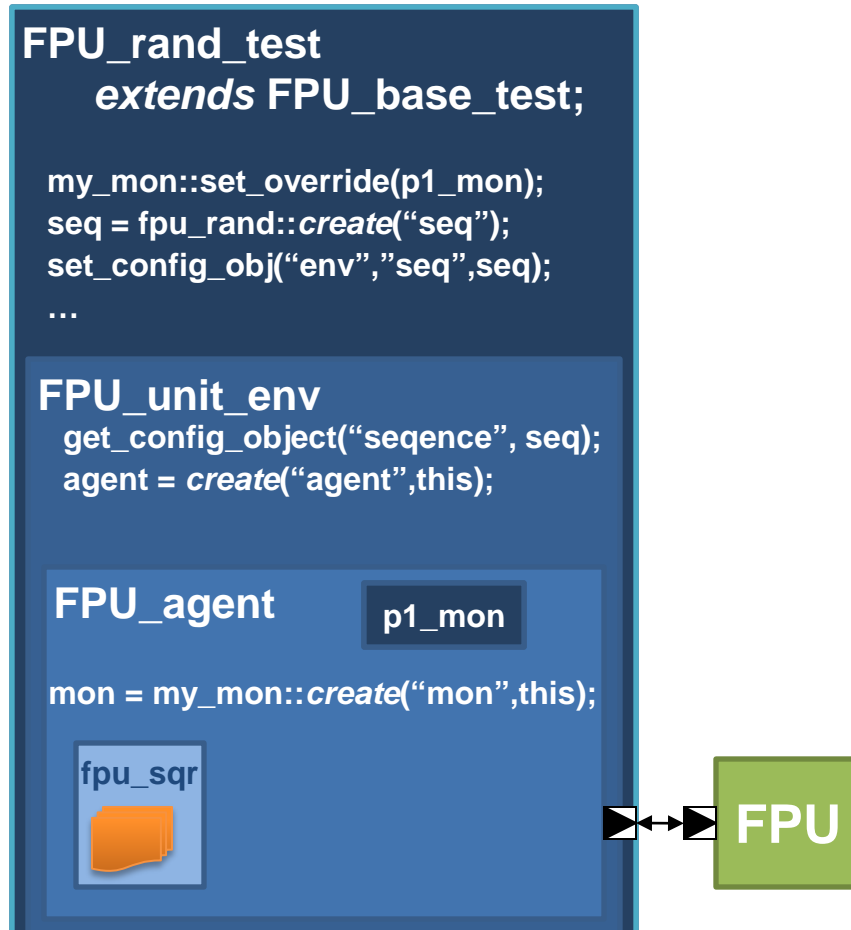
***OVM/UVM Flexibility & Modularity
Maximize Your Productivity***



OVM/UVM Test Architecture

OVM/UVM Promotes Reuse

- **Base test instantiates environment**
 - Sets default configuration
 - Configuration objects encapsulate all relevant info
- **Environment uses factory & config to instantiate children**
 - Default configurations
 - Default sequences
 - Other infrastructure
- **Components use config & factory to know what to do**
- **Extend base test**
 - Override factory settings
 - Set test-specific configurations



UVM Technical Enhancements

- **High-level methodology is OVM**
 - Reference manual and user-guide are the same
- **Various enablers were added**
 - Based on Accellera user-requirements and priorities
 - Backward compatible (see exception)
- **Areas of enhancements**
 - Callbacks
 - Built-in report catcher callback was added
 - End-of-test

UVM Technical Enhancements

■ **Callbacks**

- Allows procedural extensions to existing testbenches
- OVM2.1.1 callbacks are used
 - Type-safe callbacks
 - New additions: Type-wide and glob-style registration support

■ **Report catching callback**

- A callback to catch any message before it is emitted
- Through user-defined procedural code a user can:
 - Allow it to be issued (throw) with optional modifications
 - Prevent it from being issued (catch)
- User can register multiple catchers for the same message
- VMM had this as a function hook, we insisted on turning it to a callback

Reminder: Managing End-of-Test in OVM

- **End-of-test is the most common use of the objection mechanism**
 - Need a graceful way to end a simulation when all components have finished their planned activity
 - The built-in `ovm_test_done` objection handles this
- **Raising and dropping objections:**
 - If an `ovm_object` wants to **prevent** the test from finishing, it raises an objection:

```
ovm_test_done.raise_objection(this); // this = current object scope
```

- While this objection is outstanding, the test will not allow `global_stop_request()` to execute
- Once the object is done, it **drops** the objection

```
ovm_test_done.drop_objection(this,2); // this = current object scope
```

- Once all objections are done the test stops:



Reminder: Managing End-of-Test in OVM (Cont')

- Environments may need to allow propagation of outstanding items (DUT responses, monitoring, scoreboard activity, etc) before the test is stopped
- At every level in the hierarchy users can either:
 - Set a drain time:

```
ovm_obj_pkg::ovm_test_done.set_drain_time(this, 1000);
```

- Use all_dropped() task hook

```
class myenv extends ovm_env;  
  task all_dropped (ovm_objection objection, ovm_object  
                  source_obj, int count);  
    if(objection == uvm_test_done)  
      repeat(15) @(posedge vif.clk);  
  endtask  
endclass: myenv
```

all_dropped is recommended since it allows using provides more control and are scalable in vertical reuse as clocks or time scales change

UVM End-of-test Enhancements

■ OVM 2.1.1 End-of-test enhancements

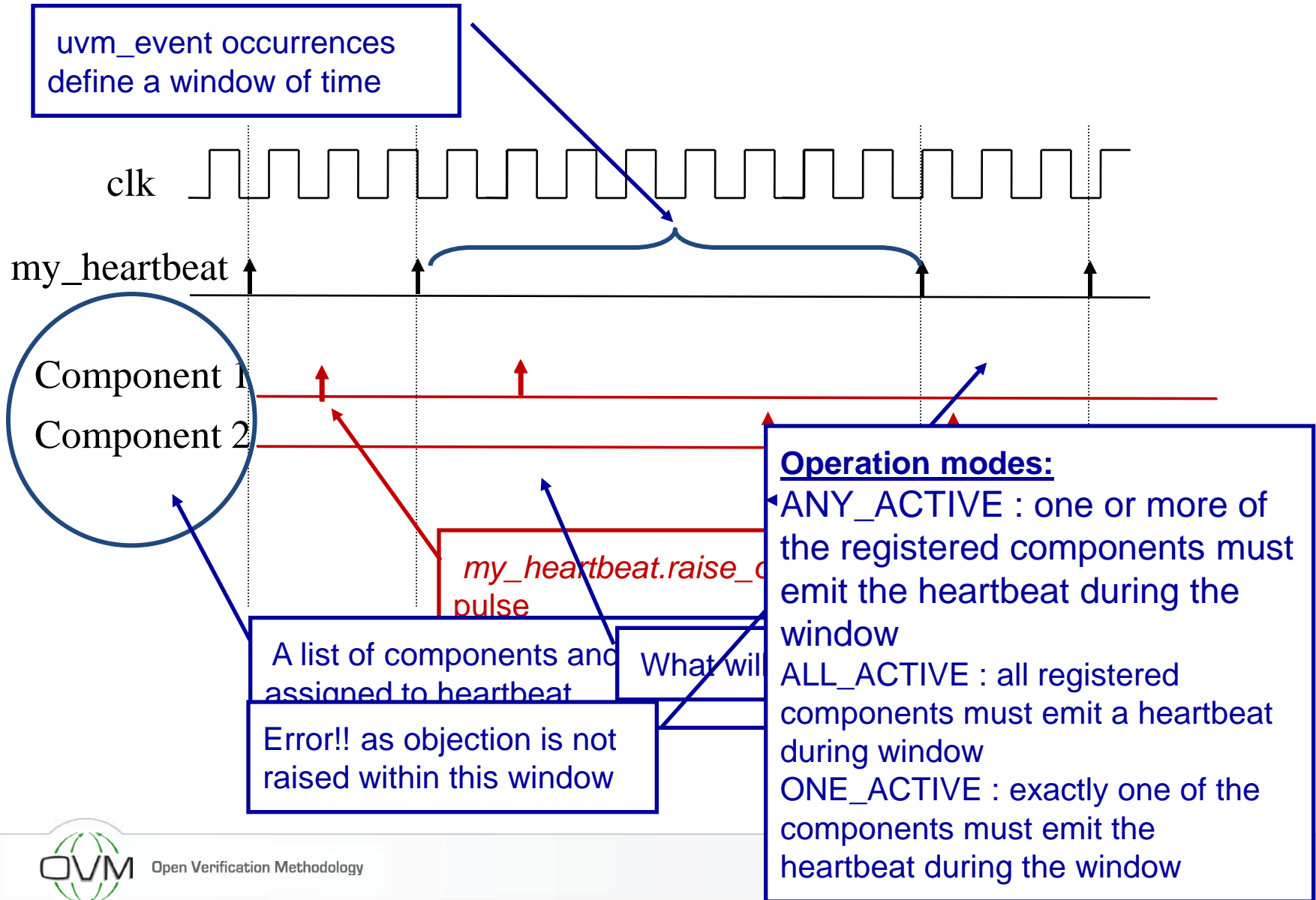
- Ability to use `set_config` to control the simulation timeout
- A user defined string description of why a specific objection is raised or dropped
 - *function void raise_objection (uvm_object obj = null, **string description = ""**, int count = 1)*
 - Note that there is minor backward compatibility issue
- Objection class has a `trace_objection` bit attached to them
 - *uvm_objection::set_trace(bit trace_objection)*

■ Heartbeat mechanism

- Determines when objects are no longer functioning properly
- Registered objects must emit an heartbeat between occurrences of a specified UVM event



Heartbeat Operation



How To Migrate from OVM to UVM?

```
class data_packet_c extends uvm_object ;  
  rand pkt_header_c header;  
  rand byte          payload [ ];  
  byte              parity;  
  rand parity_e     parity_type;
```

OVM	UVM	Rename.pl
"ovm"	→	"uvm"
"OVM"	→	"UVM"
"t1m"	→	"uvm_t1m"

```
// field declarations and automation flags  
uvm_object_utils_begin(data_packet_c)  
  `uvm_field_object( header, UVM_DEFAULT)  
  `uvm_field_array_int( payload, UVM_DEFAULT)  
  `uvm_field_int( parity, UVM_DEFAULT)  
  `uvm_field_enum( parity_e, parity_type, UVM_DEFAULT + UVM_NOCOMPARE)  
uvm_object_utils_end  
// Additional: constraints, constructor, methods  
endclass: data_packet_c
```

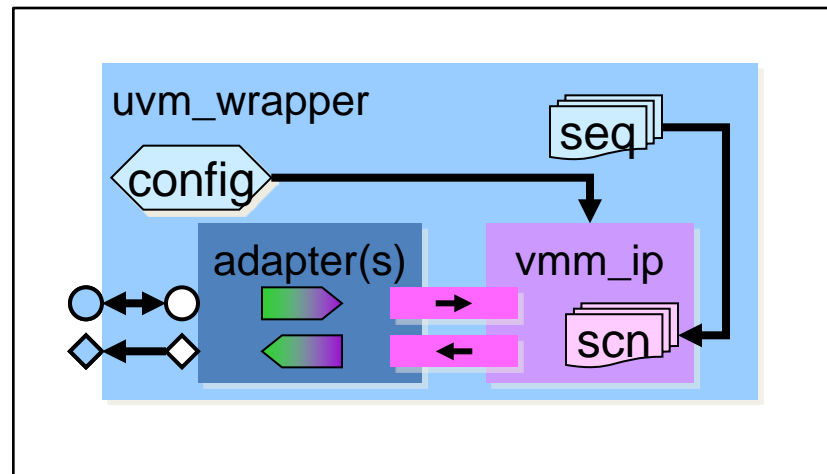
Simple scripts allows migrating OVM environments to UVM

OVM2UVM



Migration of VMM to UVM

- **Accellera OVM/VMM Interoperability layer already exists**
 - Developed for more than a year and tested on all simulators
 - Proven by users and expertise was gathered
- **Already have a VMM-2-UVM version**
 - In last stages of verifying on all simulators
 - VMM1.2 version was created to allow VMM1.2 early adopters to move to UVM
- **Migration demand has increased significantly**
 - Migrating all VMM users is likely to take time



UVM Future Enhancements

- **Registers and memory packages**
 - Requirements are being collected right now
 - The three vendors have their own solutions and one is likely to be selected as the base solution
- **Run-time Phases**
 - Coordinate component's run-time simulation without upfront planning
 - Can define domains of components that can reset together
 - Can reset an arbitrary set of compound components Need standard phases including the ability to extend the mechanism
- **TLM2.0 support**
 - Under discussion
 - Unclear what features of TLM2.0 are needed for verification
 - Multi-language support



Summary

- **OVM and UVM are officially the industry preferred methodologies**
 - High quality was demonstrated on all three simulators
 - OVM users are pleased with the UVM direction
 - Reference and user guide were maintained
- **Further development is planned in UVM**
 - Register package, phases and more
- **Migration to UVM**
 - Mentor and Cadence are committed to support OVM as long as it will be required
 - VMM testbenches can be either integrated using the interoperability package or converted
- **Thank you to the user community who supported OVM as the industry methodology, and drove us to consolidate to a single methodology – UVM!**

