De-mystifying synchronization between various verification components by employing novel UVM classes

Name : Pushpal Nautiyal
Email ID: pushpal@synopsys.com
Mobile no: +919717705649

Name : Gaurav Chugh
Email ID: cgaurav@synopsys.com
Mobile no: +919871800685

Synchronizing Verification Components

UVM class-based test benches have become as complex as the hardware under test [DUV] , and are evolving into large object oriented software designs

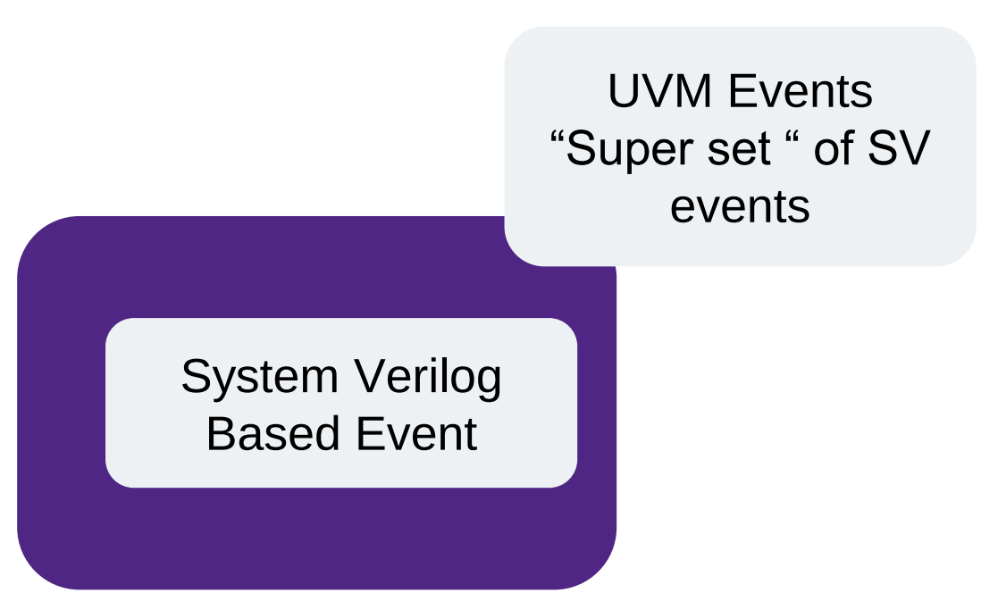
This poster is an attempt to lift the veil on sorely underutilized UVM classes such as following:

1) **uvm_event/uvm_event_pool**
2) **uvm_barrier/ uvm_barier_pool**,
3) **uvm_heartbeat**
4) **Grab/Ungrab**

UVM_EVENT/ UVM_EVENT_POOL

UVM_EVENT/ UVM_EVENT_POOL :

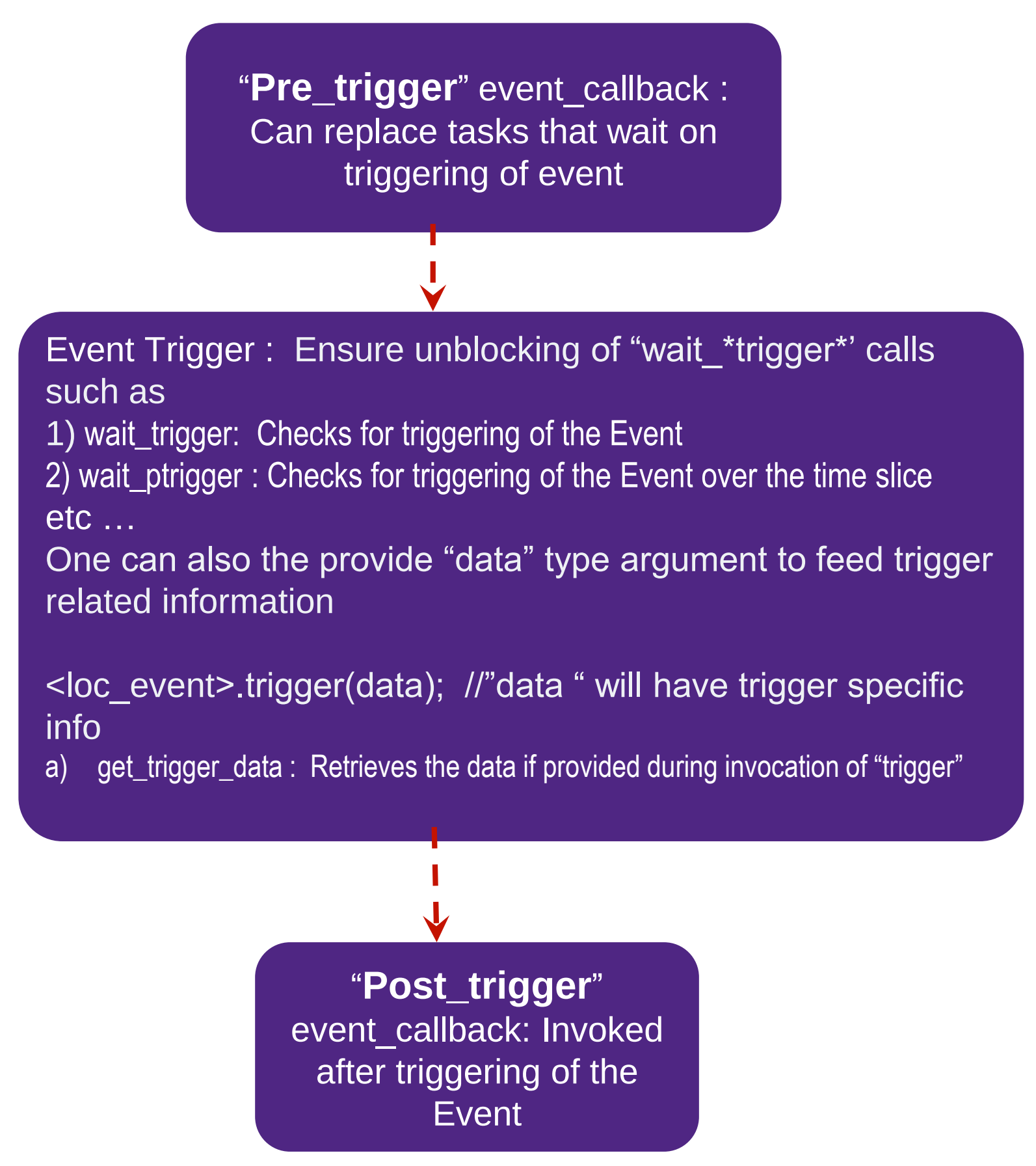
The UVM event class congregates the richness of System Verilog/Verilog events and adds a few features of its own, thus achieving leverage.


UVM Events "Super set " of SV events
System Verilog Based Event

Advantages of UVM events:

1) Provides a knob to the user just before or after the activation of uvm_events via "trigger_callbacks"
2) Can be used to return status of uvm_objects on triggering of event
3) Ease of accessibility [Event objects can be passed around the Test bench using "uvm_config_db " mechanism]

For example in verification environment two components such as "environment" and "agent" or [driver & monitor etc] may depend on single event say "reset_event" and hence can easily be shared .



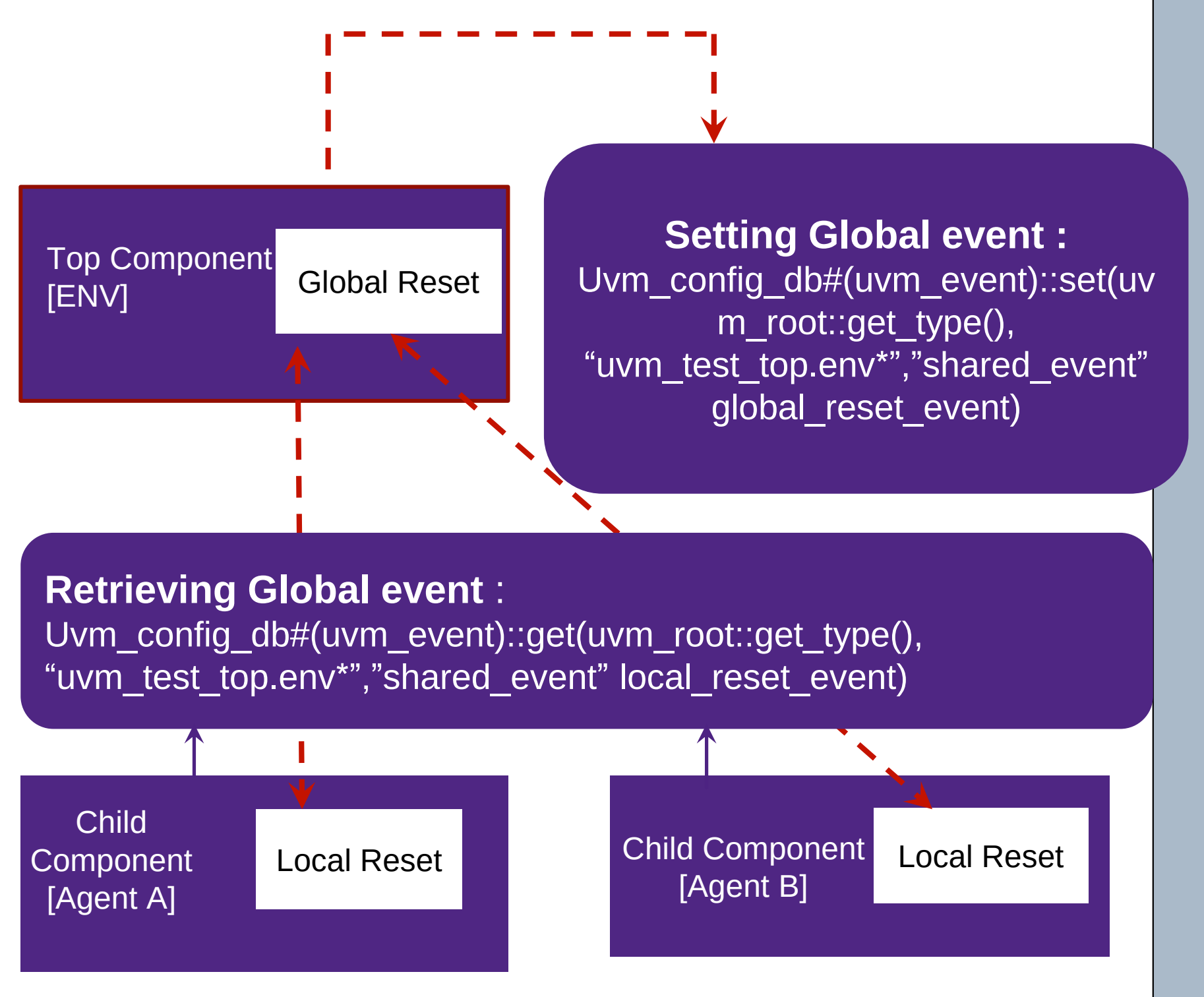
"**Pre_trigger**" event_callback : Can replace tasks that wait on triggering of event

Event Trigger : Ensure unblocking of "wait_*trigger*' calls such as
1) wait_trigger: Checks for triggering of the Event
2) wait_ptrigger : Checks for triggering of the Event over the time slice etc …
One can also the provide "data" type argument to feed trigger related information

<loc_event>.trigger(data); //"data " will have trigger specific info
a) get_trigger_data : Retrieves the data if provided during invocation of "trigger"

"**Post_trigger**" event_callback: Invoked after triggering of the Event

Synchronization of Global "Reset" Event



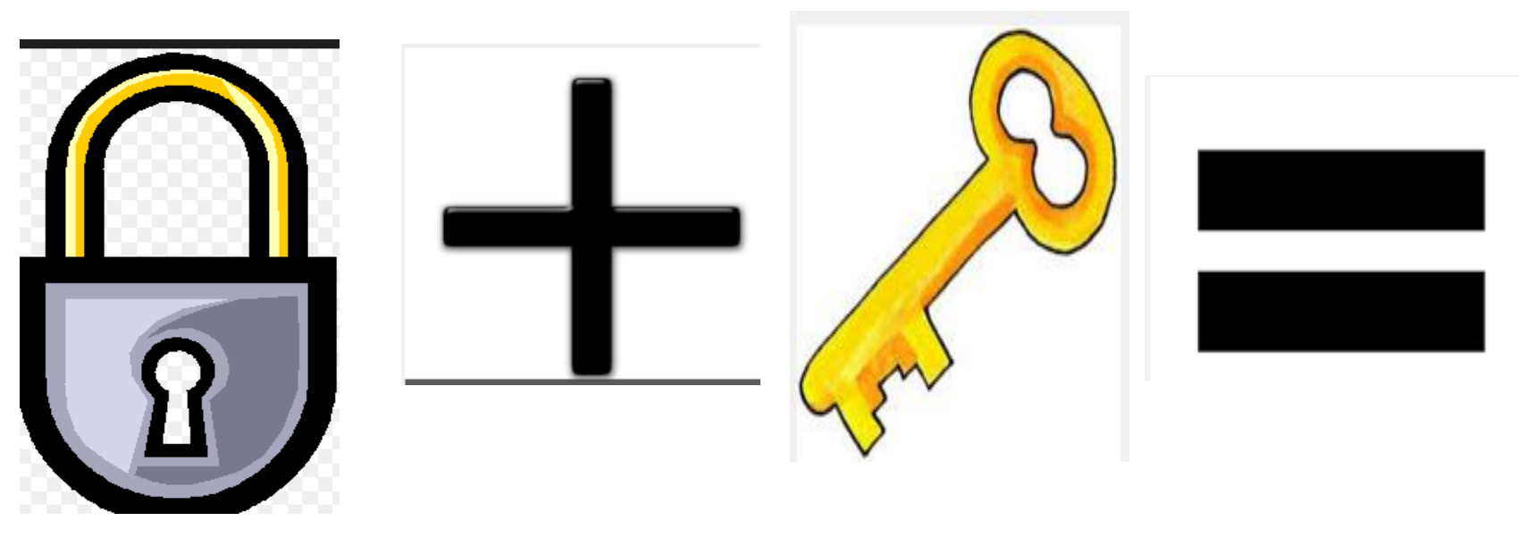
Setting Global event :
Uvm_config_db#(uvm_event)::set(uvm_root::get_type(), "uvm_test_top.env*","shared_event" global_reset_event)

Retrieving Global event :
Uvm_config_db#(uvm_event)::get(uvm_root::get_type(), "uvm_test_top.env*","shared_event" local_reset_event)

UVM BARRIER

UVM_BARRIER /UVM_BARRIER_POOL :

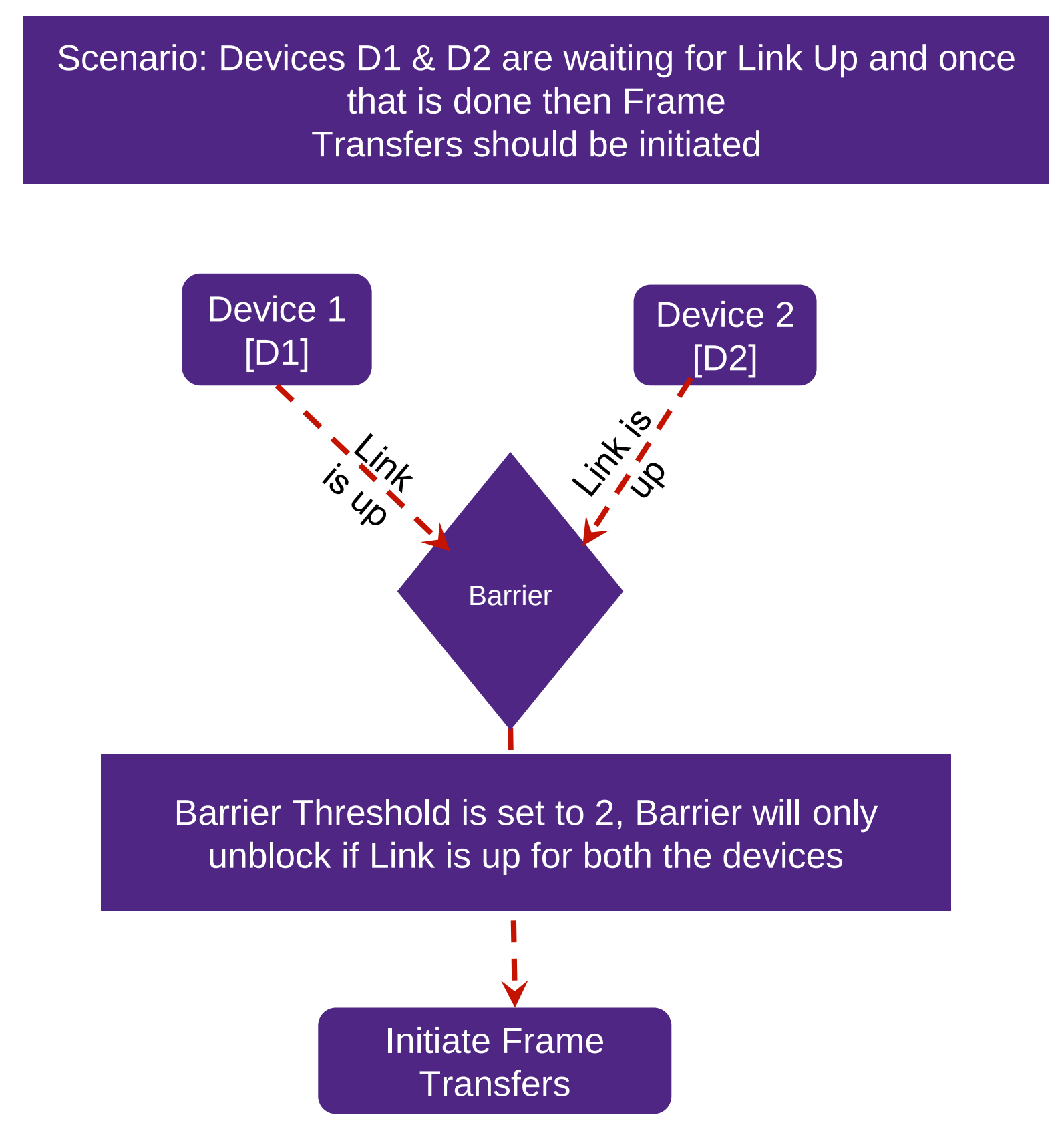
Its assets come to the fore when a user wishes to block desired number of processes until a threshold/synchronization point is achieved.
The uvm_barrier_pool classes make it easier to manage components that share the same barriers as they also can be accessed globally just as uvm_events .



One of the most active areas where it can be easily be employed is when multiple sequences are getting executed in parallel and one needs to wait for completion of all the sequences to go forward

The most prevalent API's for uvm_barrier class are :
1) wait_for : Will block until completion of processes to reach the barrier
2) set_threshold : Determines numbers of processes who would wait on the barrier before they resume
3) get_num_waiters: Queries the number of processes currently waiting at the barrier

Employing Barriers to Check for Link UP


Scenario: Devices D1 & D2 are waiting for Link Up and once that is done then Frame Transfers should be initiated
Device 1 [D1]
Device 2 [D2]
Link is up
Link is up
Barrier
Barrier Threshold is set to 2, Barrier will only unblock if Link is up for both the devices
Initiate Frame Transfers

UVM HEARTBEAT

UVM_HEARTBEAT :

The UVM Heartbeat behaves as a watchdog timer and is quite powerful. It watches for activity in the test bench components and if it finds that there isn't the right amount of activity in that window - will issue a fatal message and end the simulation. This can catch a simulation lock-up early on – even before the global timeout kicks in, potentially saving a significant amount of time.

Here are few of the scenarios of interest in ceasing a simulation rather than UVM_TIMEOUT being called out :
1) Absent Connections between the Verification Components
2) Simulation Hang in State Machines

To employ the UVM heart beat we need to associate a specific objection object , and the heartbeat object must raise (or drop) the synchronizing objection during the heartbeat window

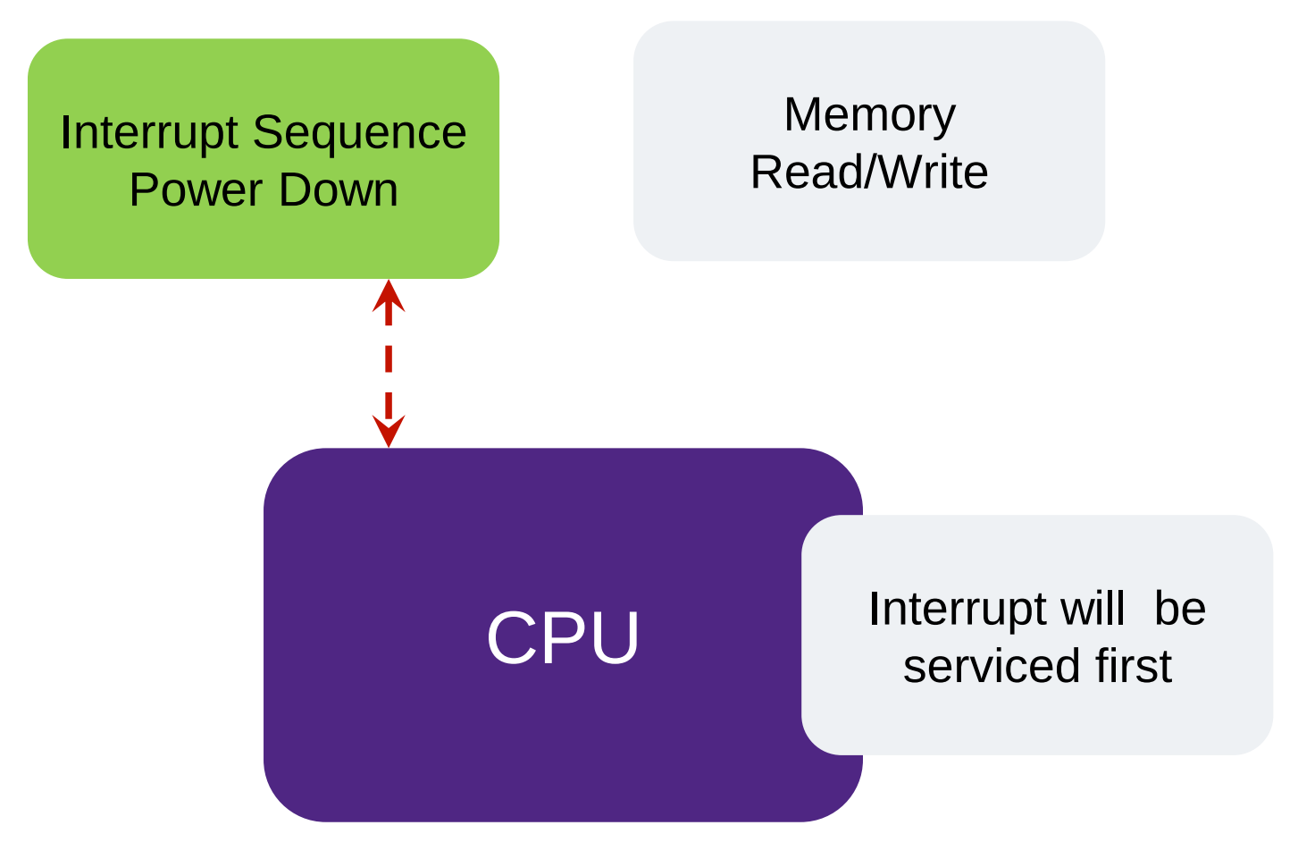
The most prevalent API's for uvm_hearbeat class are :
1) set_mode: Sets or retrieves the heartbeat mode:
The heartbeat can be configured so that all components (UVM_ALL_ACTIVE), exactly one (UVM_ONE_ACTIVE), or any component (UVM_ANY_ACTIVE) must trigger the objection in order to satisfy the heartbeat condition.
2) Add/remove: Add/Removes a single component to the set of components to be monitored
3) set_heartbeat: Sets the target list of components that are required to be monitored and setting of the heartbeat event. Soon after invocation of this monitoring is initiated

GRAB /UNGRAB

GRAB /UNGRAB :

This mechanism provides the sequence with exclusive access to the driver and will allow a sequence to complete its operation without any other sequence operations in between them . The grab() method requests a lock on the specified sequencer. A grab() request is put in front of the arbitration queue. It will be arbitrated before any other requests.

This mechanism is highly recommended in scenarios when a virtual sequencer requires full control over its sub/child sequencers for a limited time and then lets the original sequences continue working.
From an implementation perspective it will be highly useful in generating INTERRUPT Sequences where disabling of sub/child sequencers is required and highest priority needs to be given to specific sequences


Interrupt Sequence Power Down
Memory Read/Write
CPU
Interrupt will be serviced first

Psuedo Code :

virtual task body();
// **Grab the cpu sequencer if not virtual**.
if (sequencer.cpu_seqr != null)
p_sequencer.cpu_seqr.grab(this);

// **Execute a sequence.**
`uvm_do_on(intrpt_seq,sequencer.cpu_seqr)

// **Ungrab**.
if (sequencer.cpu_seqr != null)
sequencer.cpu_seqr.ungrab(this);