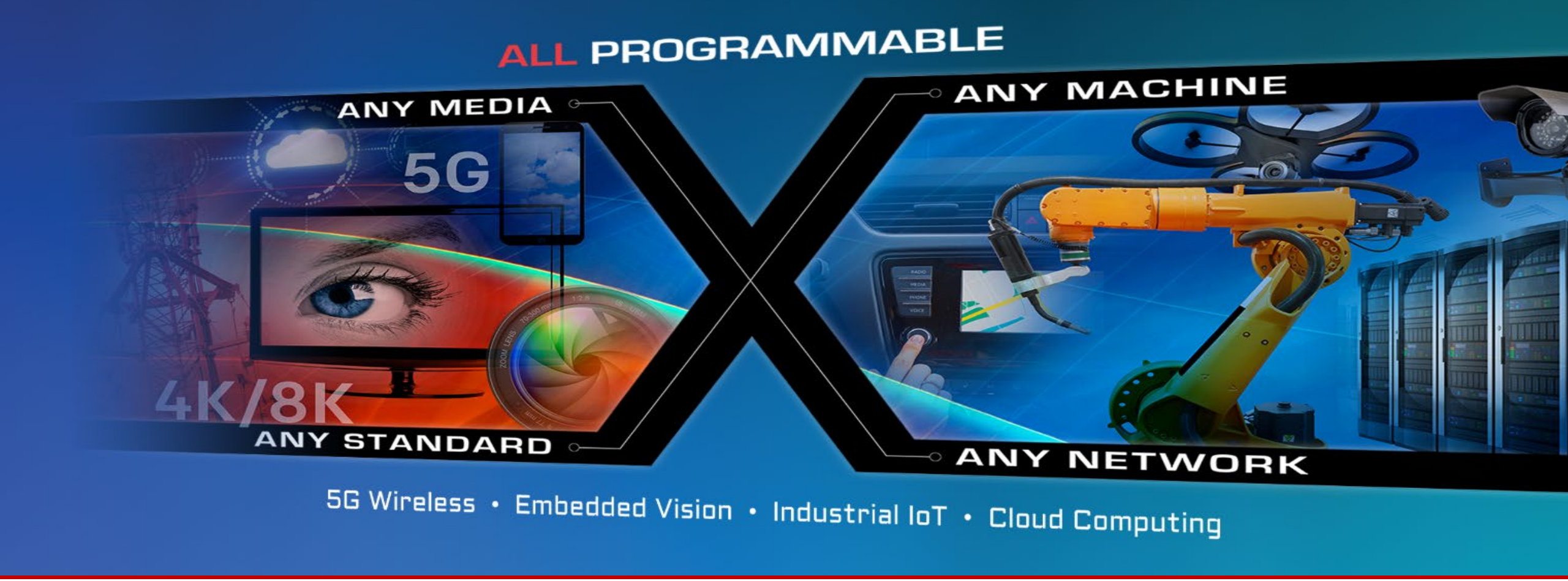
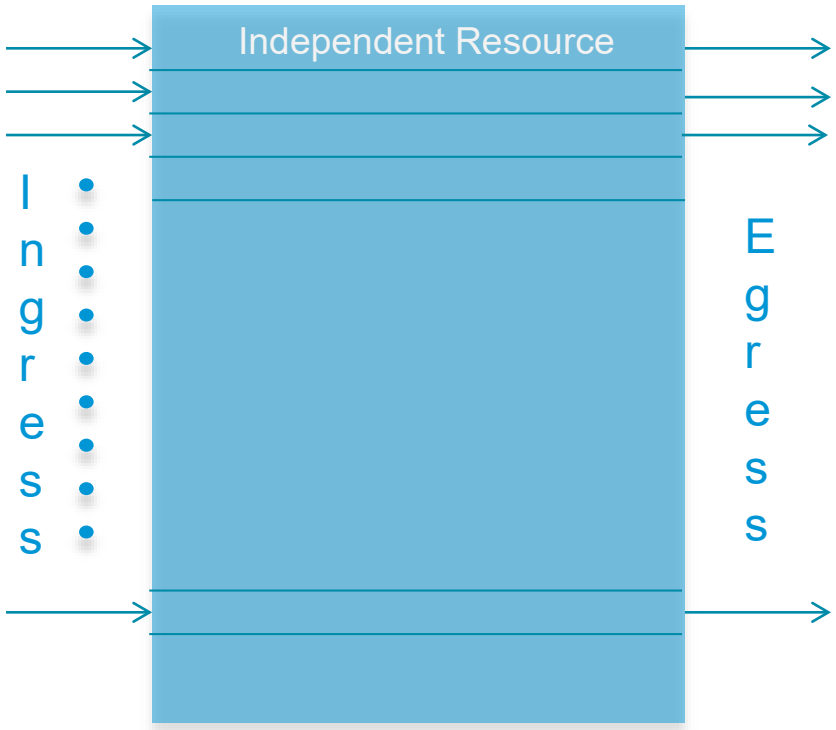


XILINX
ALL PROGRAMMABLE™

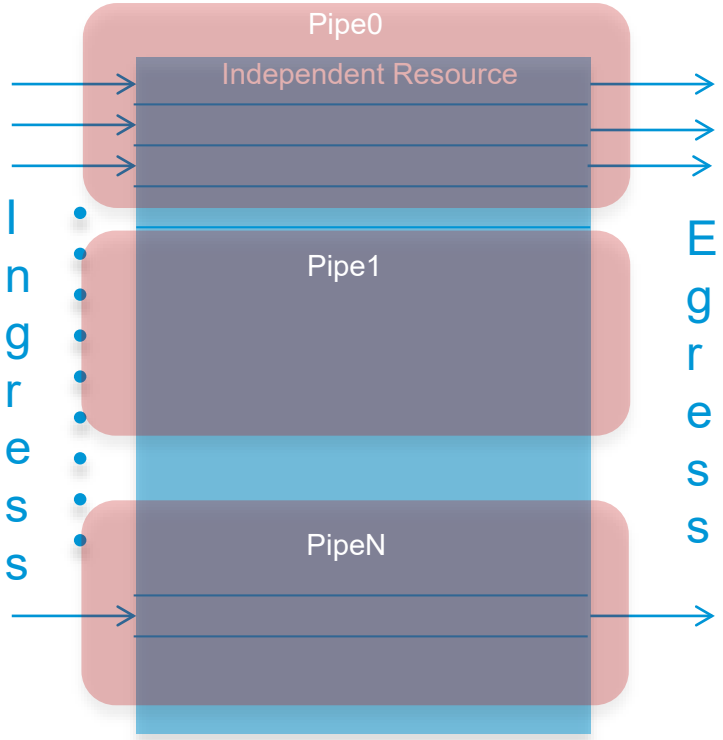
A novel approach to create multiple domain based DV architecture to address typical Verification challenges, for the DUT with mutual exclusive functionalities, using UVM Domains

What's Mutual Exclusivity in DUT/ASICs?

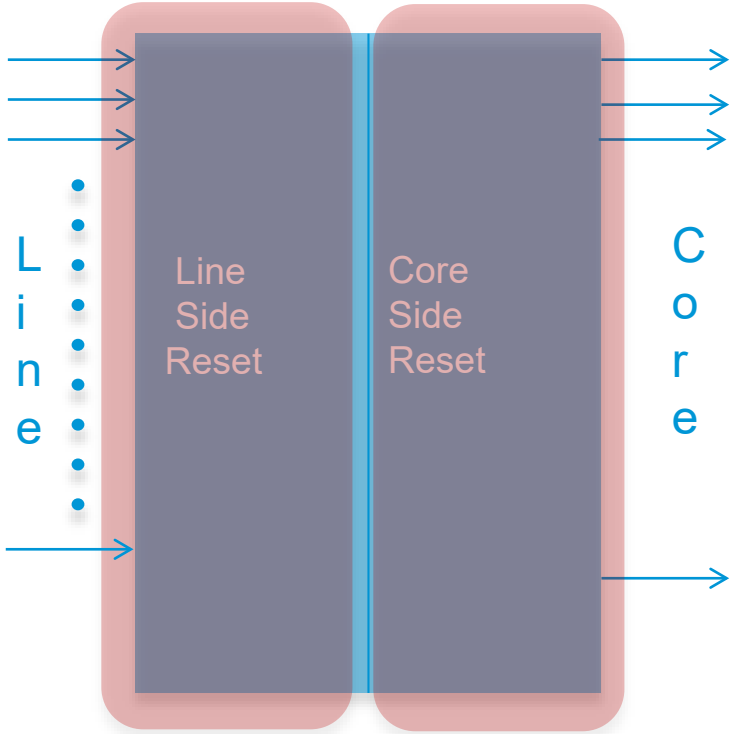
- The logical partitioning in a DUT where
  - the functionality of one partition is mutually exclusive to other partitions.
  - Each logical partition is part of separate Reset or Clock domains



Multiple Independent ports



Multiple Independent pipes



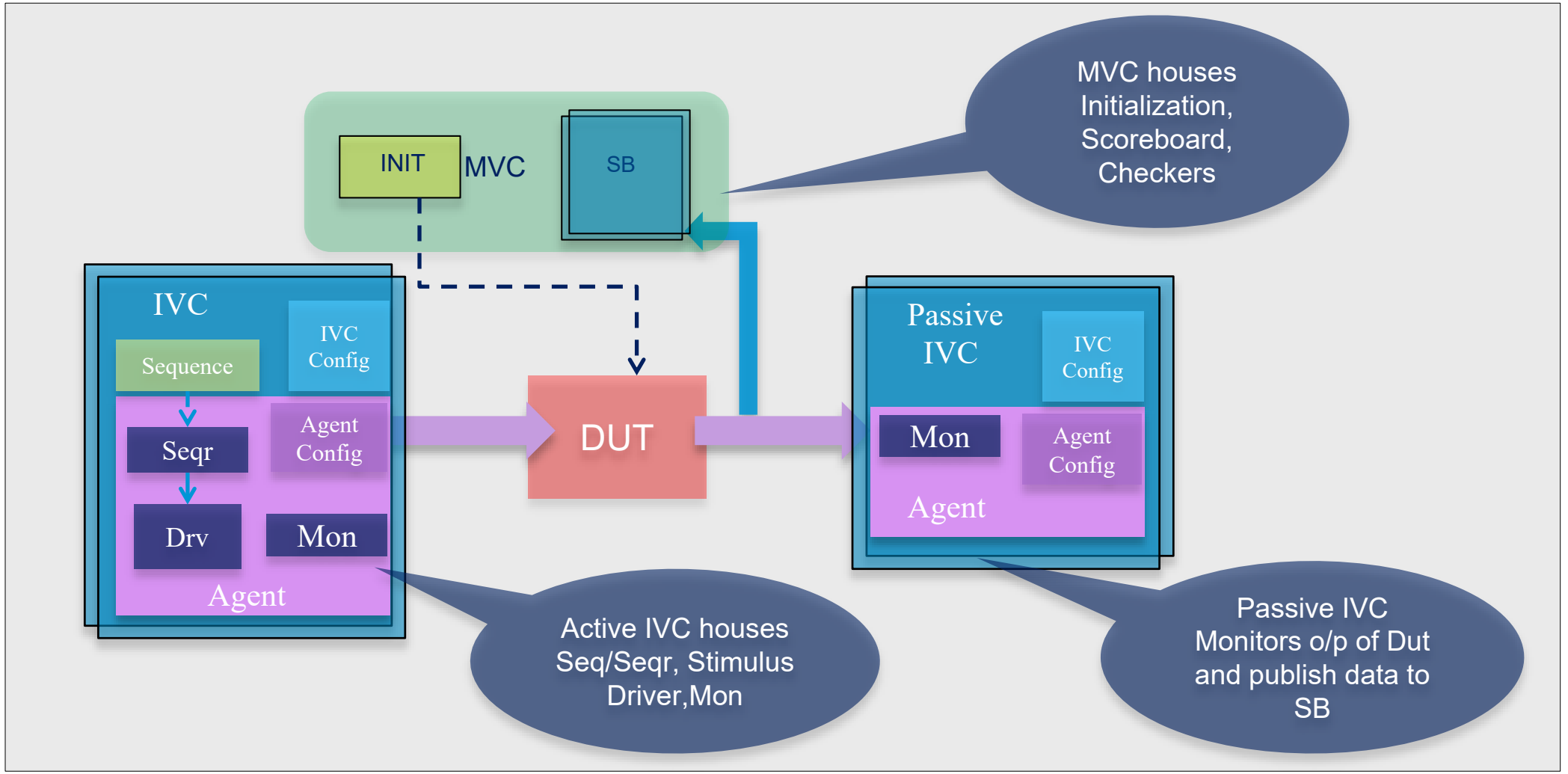
Multiple Reset Domains

DV Challenges

- DV Environment should have the similar mutual exclusivity that RTL has.
- A subset of the environment will go through changes along with a subset of RTL
  - Rest of the RTL and DV should be running without any interruptions.
- The traffic generation and data integrity checks should be aware of these transitions
  - Components need to fine tune their activity.
- The environment should be able to handle these transitions dynamically, at any point within simulation.
- Directed approach is simply too much work
  - mainly due to the size of the DV environment, and
  - the dynamic nature of the DUT activity

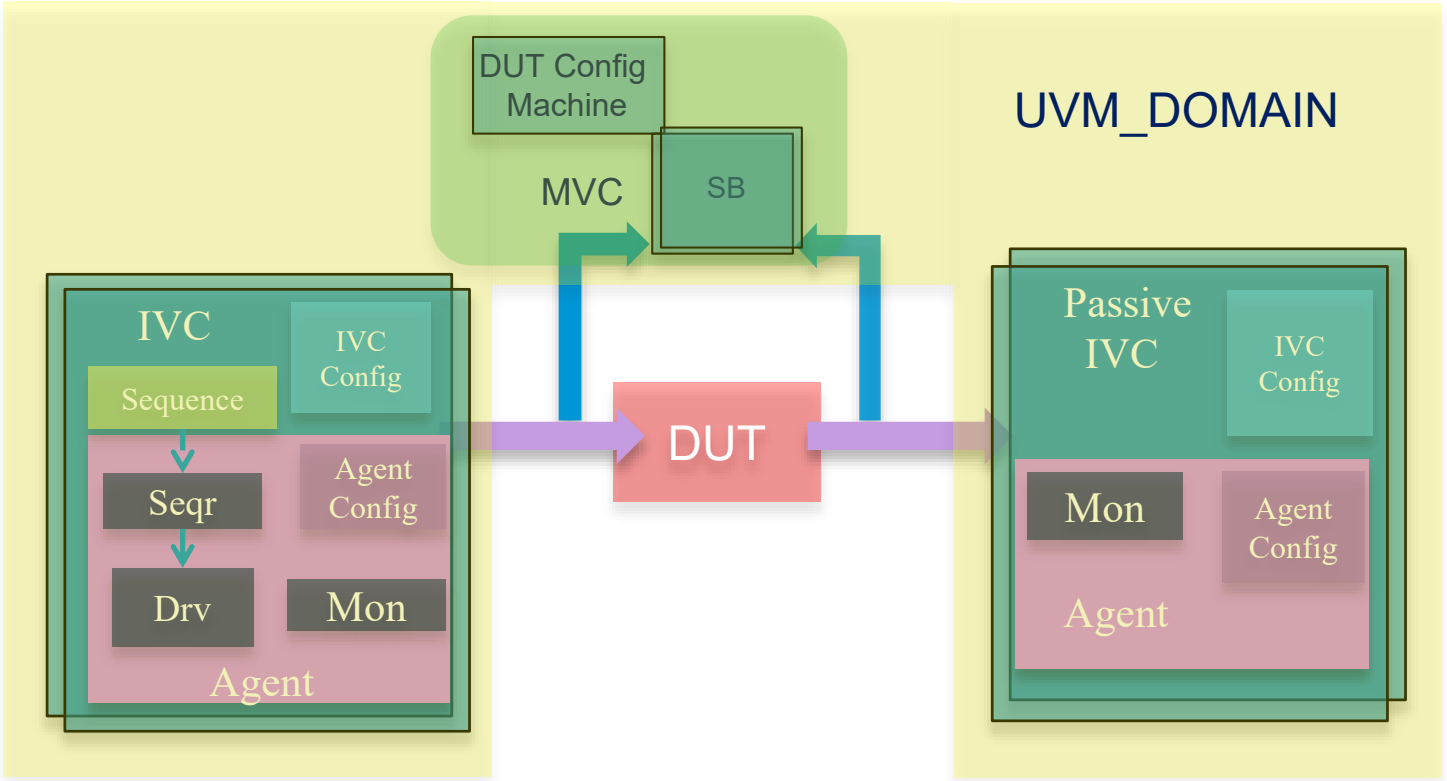
UVM_Domain , Advantages & Usage

- Typical DV Environment in UVM



UVM_Domain , Advantages & Usage Contd…

- By default there's only one domain, named 'uvm_domain'
- uvm_domain is a class, extended from uvm_phase class.
- This class is responsible for controlling the phasing mechanism among all the DV components.



Single Domain DV Architecture

UVM_Domain , Advantages & Usage

- For mutual exclusivity different uvm_domain can be created as follows:

```
uvm_domain m_usr_domain[N];

foreach (m_usr_domain[i])
    m_usr_domain[i] = new
($sformatf("m_usr_domain[%0d]",i);
```

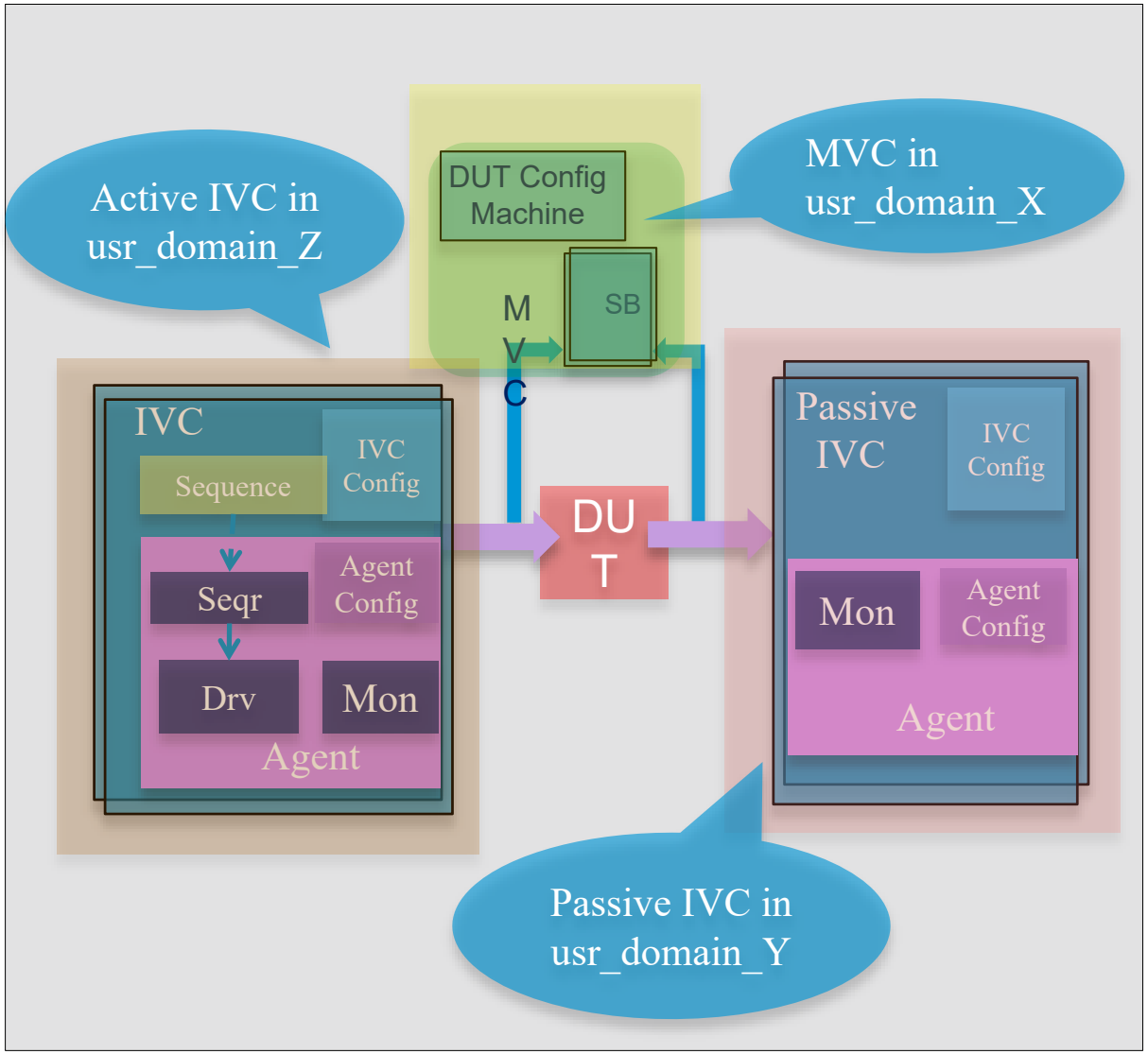
- Each domain brings in a separate phase scheduler which are completely asynchronous to each other (unless configured otherwise).
- Map the DV components, based on implementation

Recursive:

```
foreach (m_usr_ivc[i])
    m_usr_ivc[i].set_domain(m_usr_domain[i],1);
```

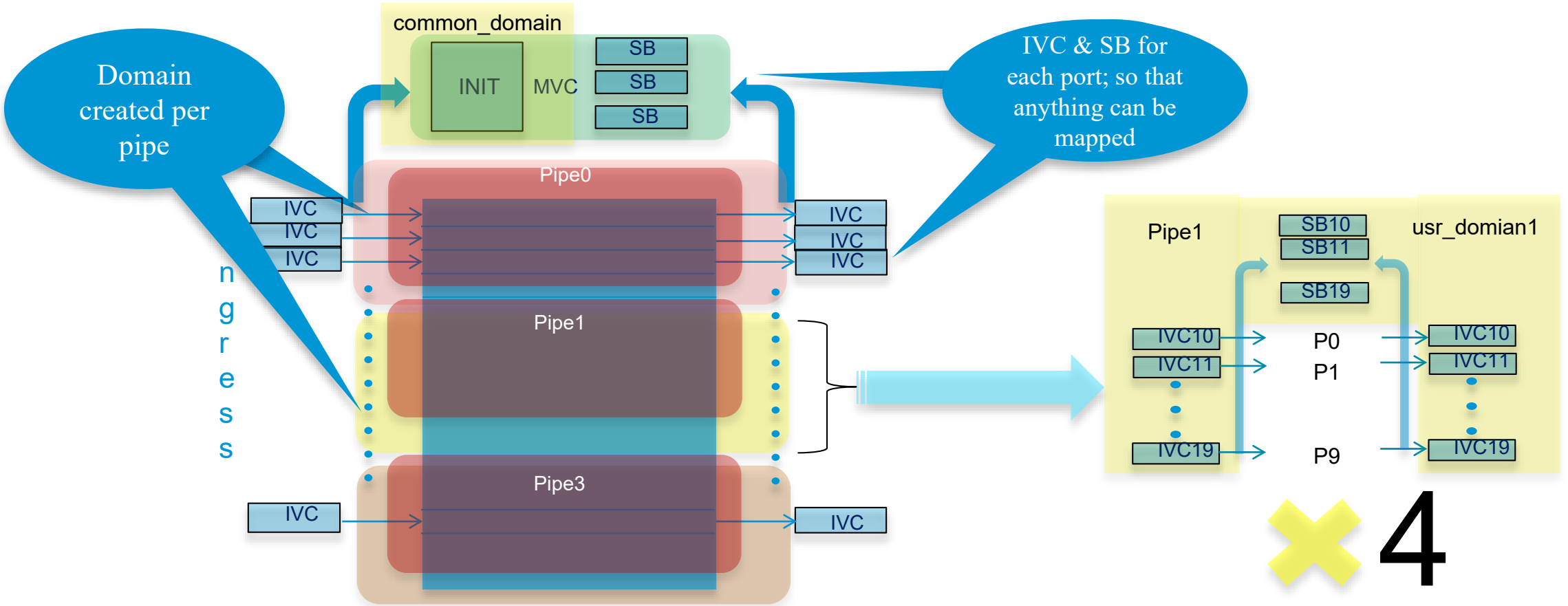
Non-Recursive:

```
foreach (m_usr_ivc[i])
    m_usr_ivc[i].set_domain(m_usr_domain[i],0);
```



Concept Realization: Ethernet Pipe Verification

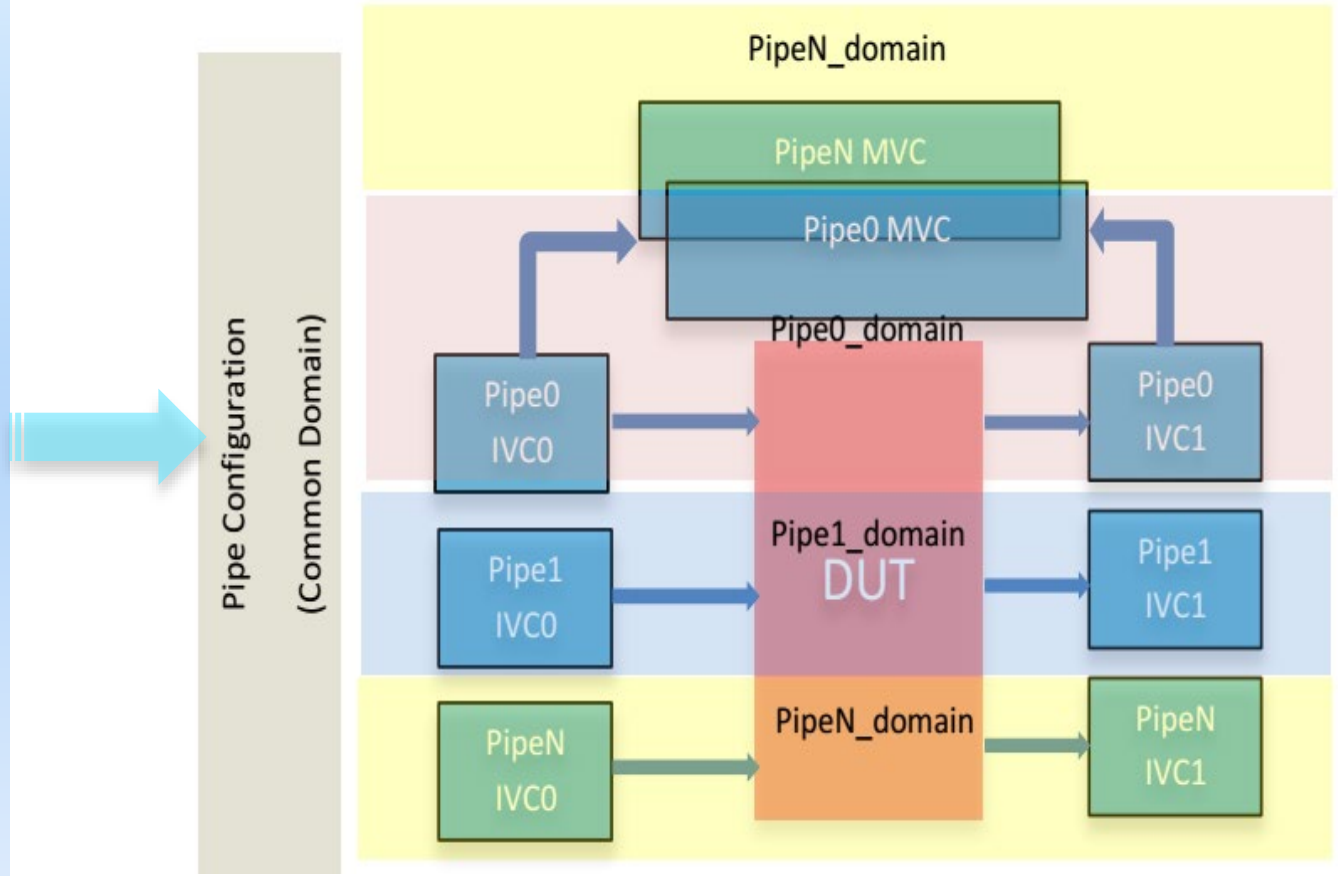
- Sample DUT has four mutually exclusive pipes each having 10 Ethernet ports at max.
- Each Pipe can be configured in 10GE, 100GE or 40GE mode independently
- Within a pipe two ports no mutual exclusivity for this DUT. [Just an example]



Concept Realization: Top Level DV Architecture

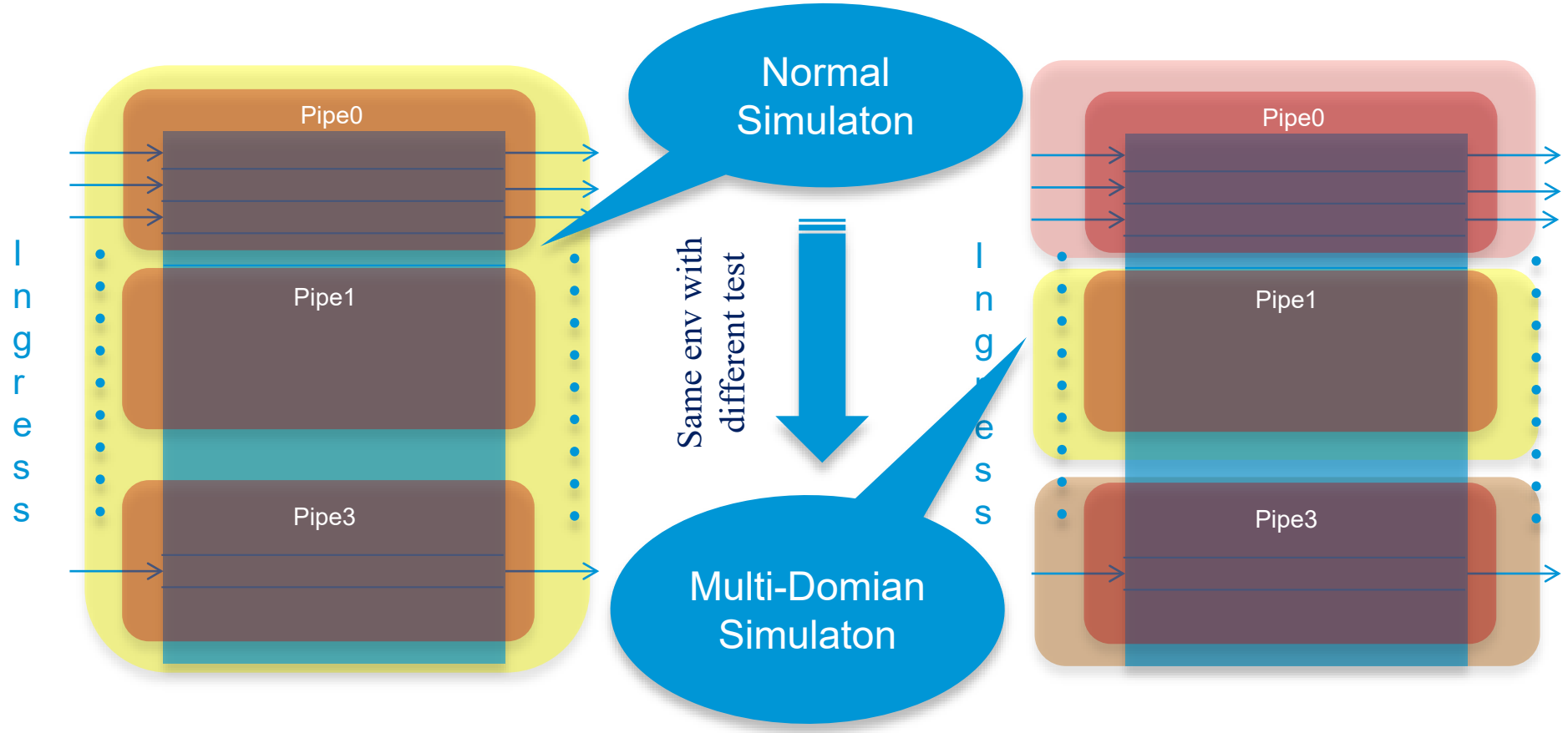
- Test-Layer, is chosen to create and map DV components with corresponding domains.
- This gives an added advantage of adding DOMAIN-based infrastructure to a already matured DV environment
- We can also run single domain tests, simultaneously with multidomain tests.

```
class user_test extends uvm_test;
  …..
  uvm_domain m_pipe_domain[N];
  uvm_domain m_common_domain;

  function new(string name, uvm_parent parent);
     ……
     foreach (m_pipe_domain[i])
        m_pipe_domain[i] = new ($sformatf("m_pipe_domain[%0d]",i);
     m_common_domain = new ("m_common_domain");
  endfunction
 ….
  function void connect_phase (uvm_phase phase)
     ……
     //Recursive:
     foreach (m_pipe_ivc[i])
       m_pipe_ivc[i].set_domain(m_pipe_domain[i],1);
     foreach (m_pipe_mvc[i])
       m_pipe_mvc[i]. set_domain(m_pipe_domain[i],1);
      m_config.set_domain(m_common_domain)
endfunction

endclass
```



Concept Realization: DV Architecture

- Choosing Test-Layer to create and map the domains have following advantages:
  - This gives an added advantage of adding DOMAIN-based infrastructure to a already matured DV environment
  - From Test-Layer all the DV components are hierarchically visible, so mapping can be done seamlessly
  - We can also run single domain tests, simultaneously with multi-domain tests.

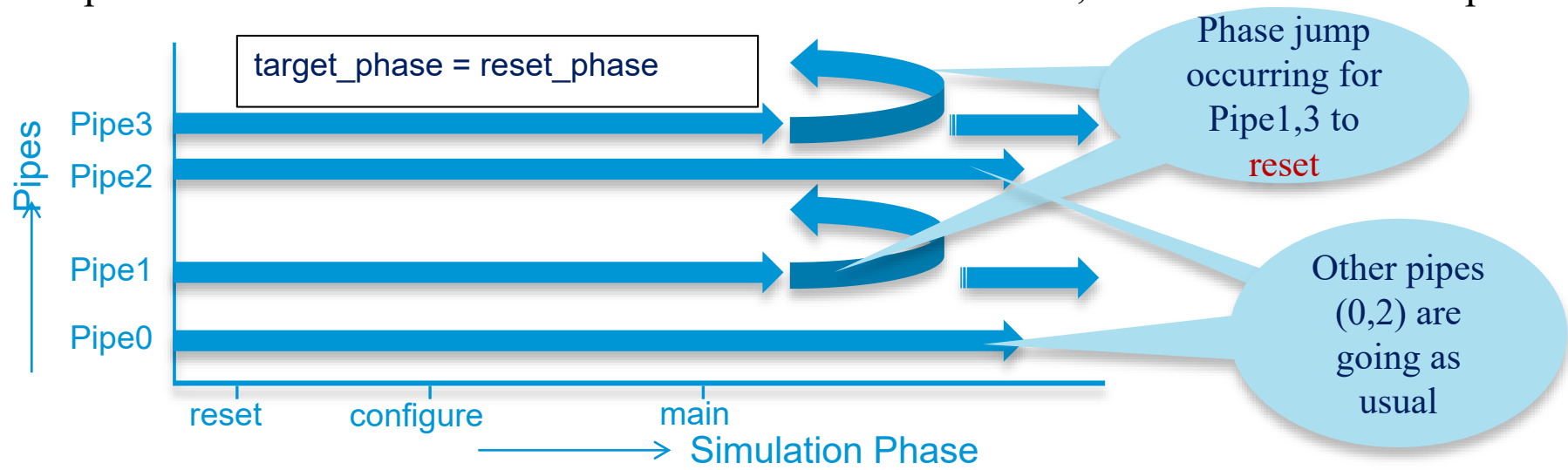


Concept Realization: Simulation Phasing

- Selected uvm_domains are made to do the phase jumping as shown below:

```
class user_test extends uvm_test;
   ....
   function void main_phase (uvm_phase phase)
     ……
     foreach (m_pipe_domain[i]) begin
     //If pipe 'i' is getting reconfigured
     if (reconfired_pipe[i]) begin
       m_pipe_domain[i].jump (target_phase::get());
     end
     m_common_domain.jump (target_phase::get());
   endfunction
……
endclass
```

- DV components from different domain will behave as shown below, as a result of selected phase jump

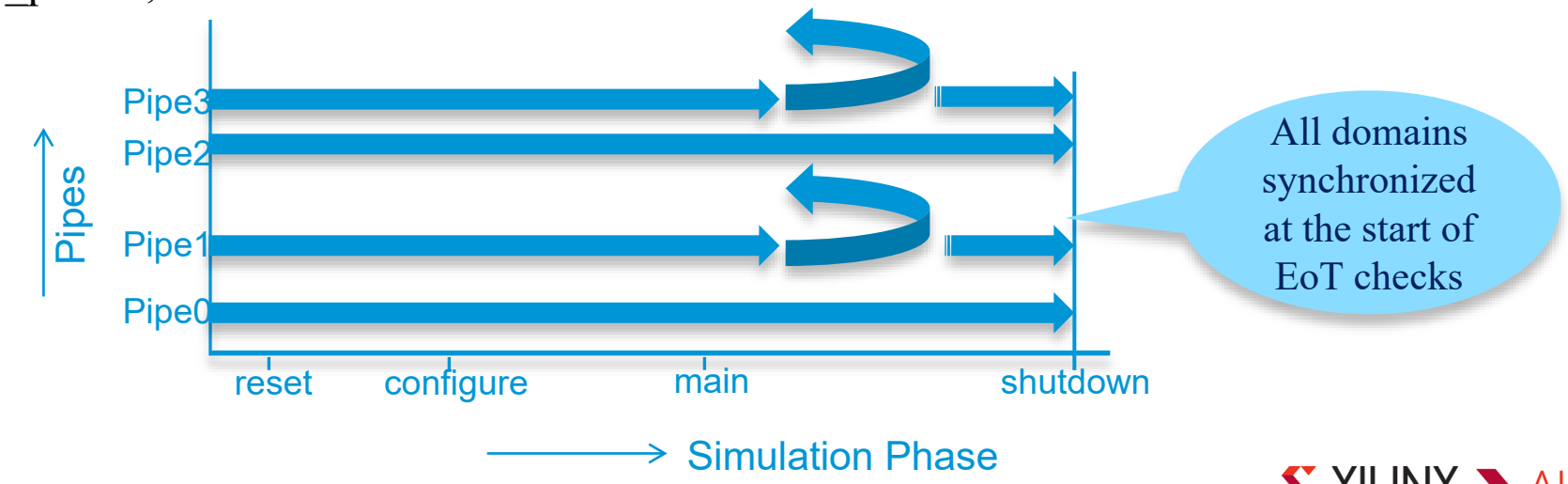


Concept Realization: Multi Domain Synchronization

- If needed domains can be synchronized again with respect to a particular phase as follows:

```
class user_test extends uvm_test;
  …..
  function void connect_phase (uvm_phase phase)
    ……
     foreach (m_usr_domain[i]) begin
       m_usr_common.sync (m_usr_domain[i]);
    end
    //NOTE: There's unsync API as well.

  endfunction
   ……
endclass
```

- One such scenario is when End-Of-Test checks will start, user might want to synchronize all the domains at 'shutdown_phase', as shown below



Concept Realization: Results & Analysis

- Once the environment is ready and loaded with all these flexibilities, it's very straight forward to achieve all possible mode transitions for one or more randomly selected pipes, in a fully automated way.
- Four tests covered almost 400 odd unique mode transitions in a multiple Pipe Ethernet DUT, leveraging this environment.
- Almost 120 odd DV components were handled in fully automated fashion during these transitions.
- More common usages of multi-domain DV architecture are, the designs with multiple ResetDomains.
    - For example, Ethernet framer (MAC+PCS) and the core logic (TCAM look-up, Buff Manager, Queue Manager etc.) will be in two different reset domain
- Reset and recovery of each domain is mutually exclusive and multi-domain DV environment can ease our effort in verifying these
- Short-comings of multi-domain DV architecture
    - It will increase to overall complexity of the test execution.
    - Testbench designers need to be very careful while integrating the flow, otherwise user might end up debugging a lot of hang issues, causing from testbench domains.
    - Testbenches which doesn't use UVM runtime phases and only relies on 'run_phase', will not get much benefit out of this methodology.
    - Memory footprint gets increased