Global Broadcast with UVM Custom Phasing

Jeremy Ridgeway, Dolly Mehta

Avago Technologies, Inc.

Agenda

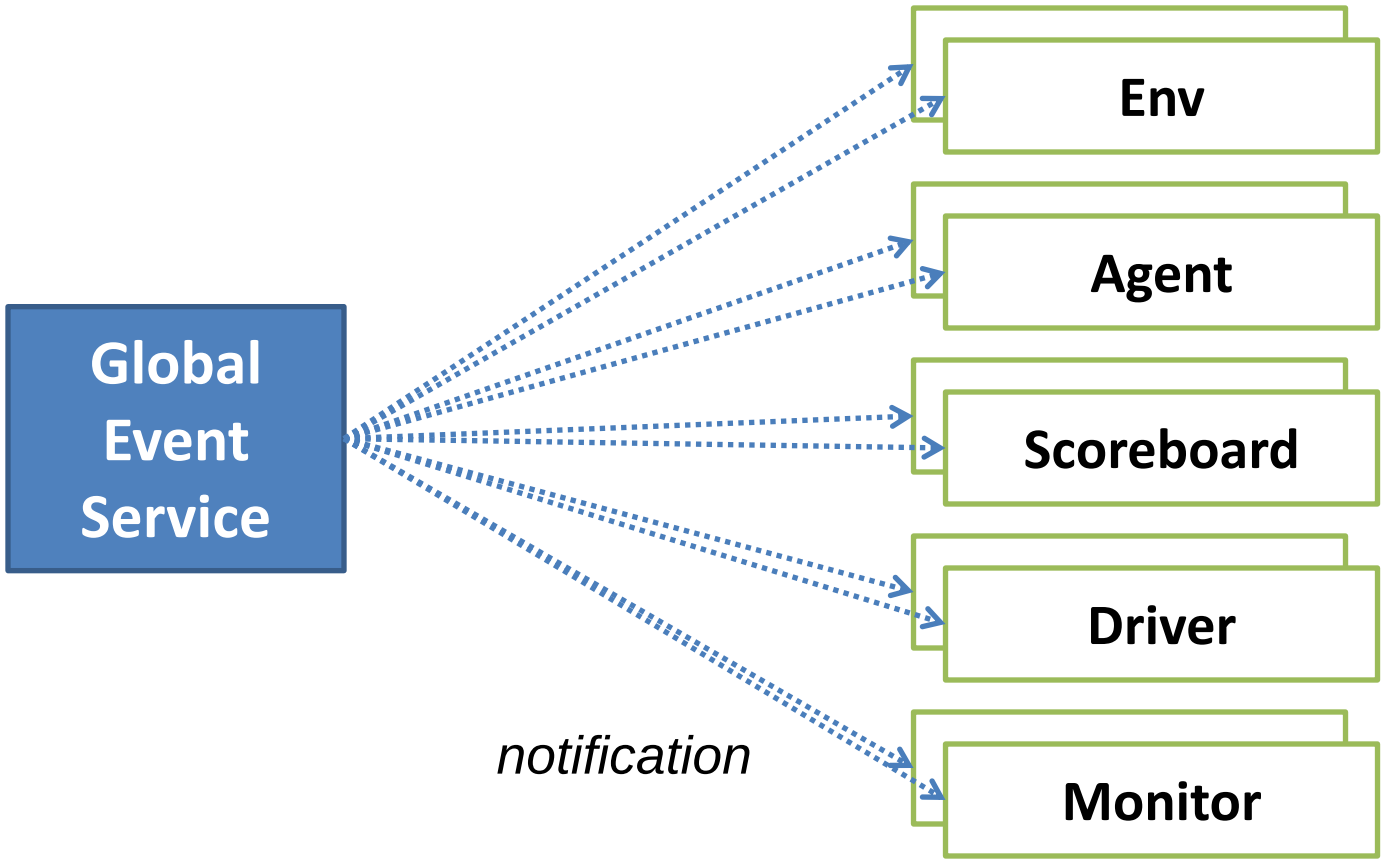
- Why phasing
- Custom Phasing Architecture
- Custom Phasing Framework
- Master Component

Goals

- DUT/Environment Dynamic Hard Reset
- DUT/Environment Dynamic Re-configuration
- Environment Quiescent State

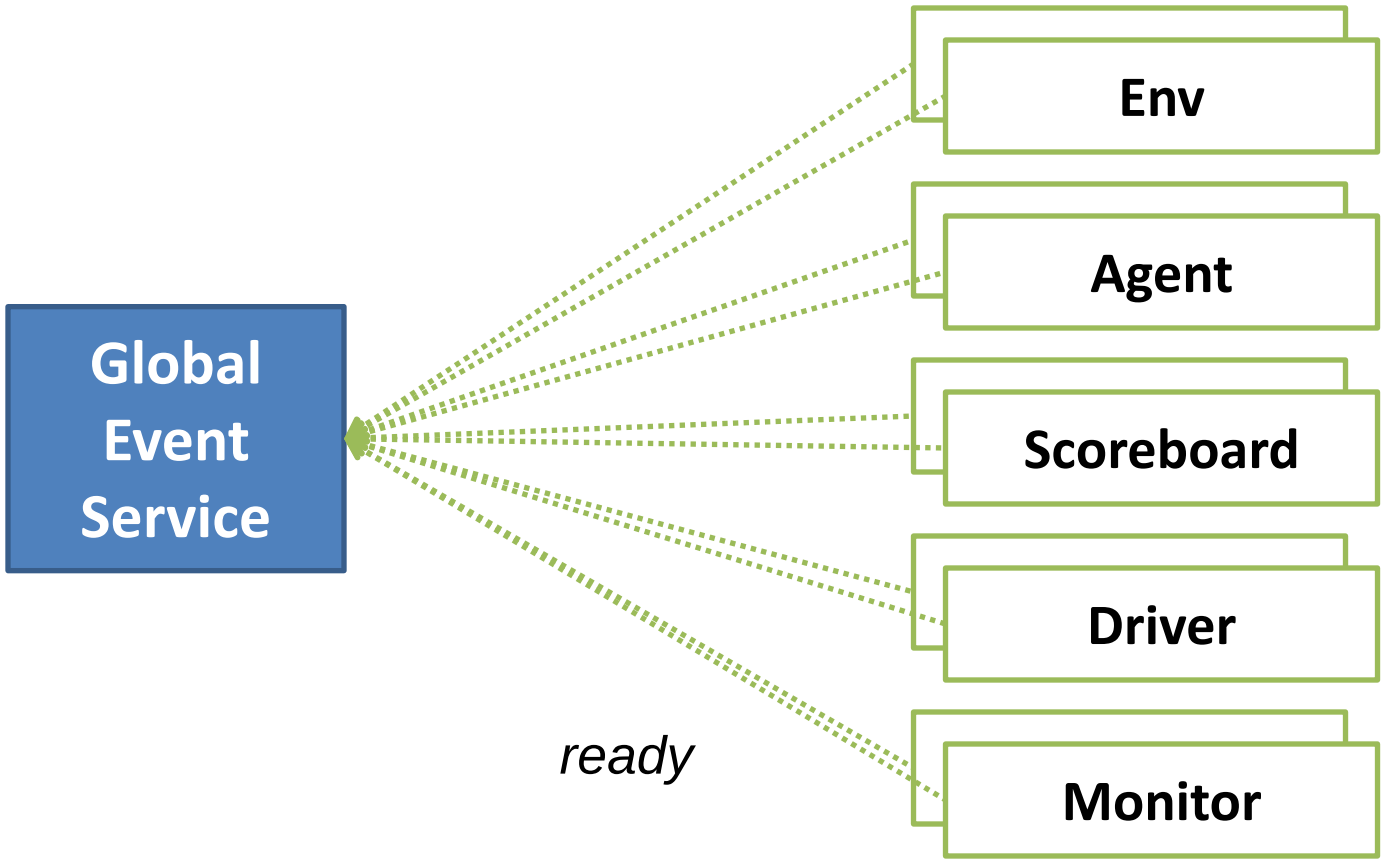
- Warn components before event happens
- Initiate event only when environment is ready

Notify Components Before Event



- One service to notify all environment components

Wait for Components to be Ready



- Initiate event after environment is ready

Solution: Phasing!

- Already connected with all components
- Extensible for custom phases

- Centralized control:
  - Provides notification
  - Easy to wait for ready

- Passive
- Nothing new to learn

Agenda

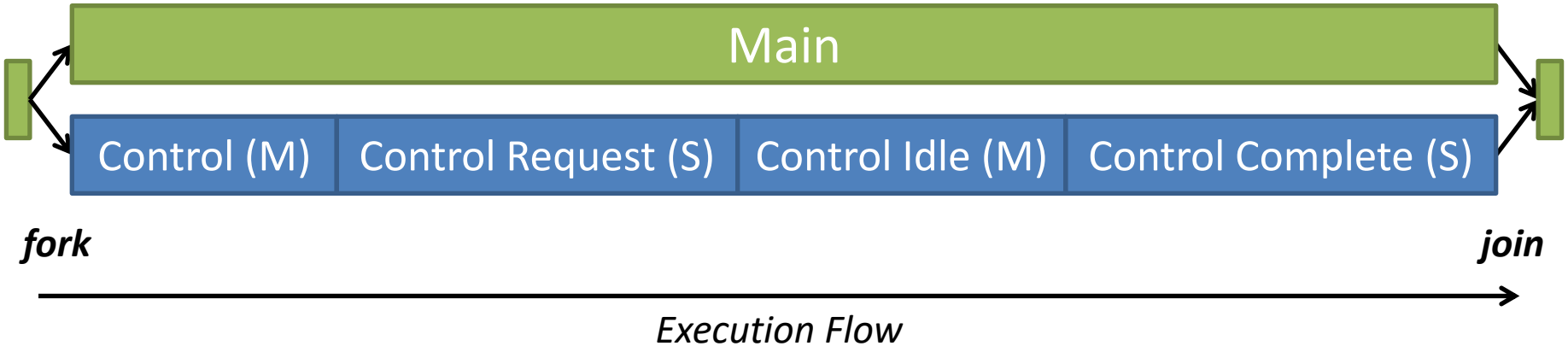
- Why phasing
- Custom Phasing Architecture
- Custom Phasing Framework
- Master Component

Run-time Phase Schedule

UVM	Reset	Configure	Main	Shutdown
Action	DUT Hard Reset	DUT Initial Config	Data Traffic	Checking

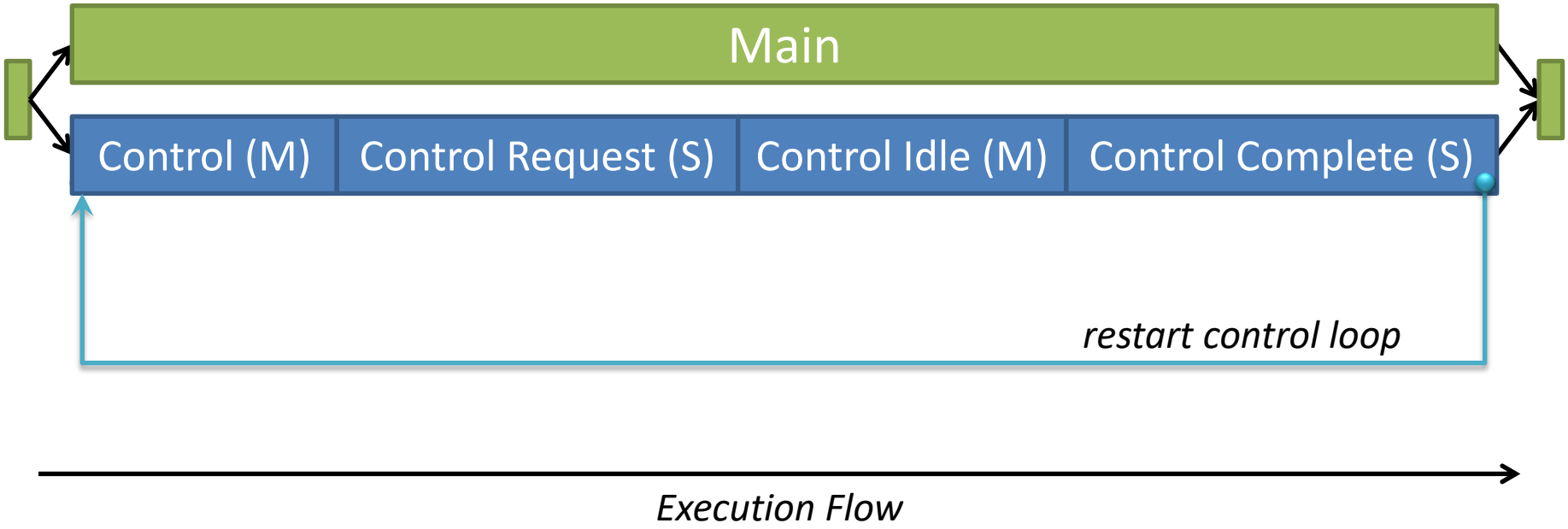
- Quiescent state in Main Phase only
- Hard reset jump from Main to Reset Phase
  - Set custom phase schedule in parallel with Main Phase

Custom Phase Schedule Master/Slave



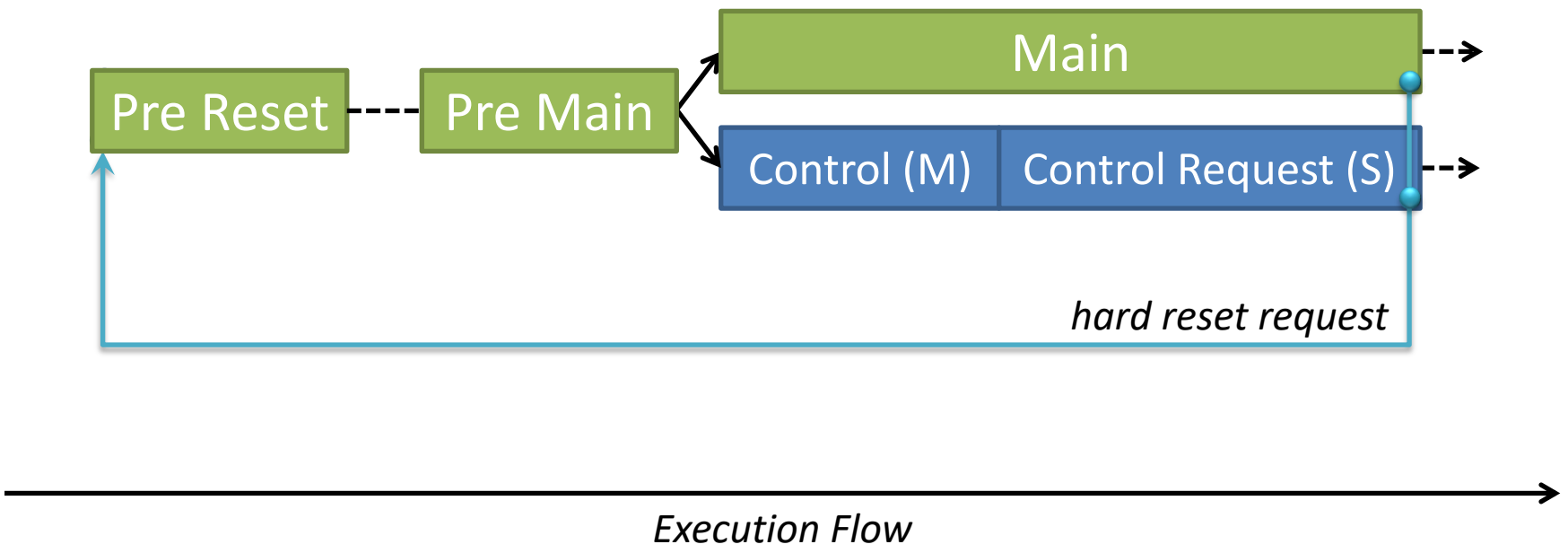
- Control – Master only; handles event requests
- Control Request – Slaves; preparation for event
- Control Idle – Master; Quiescent State
- Control Complete – Slaves; restart stimulus data

Custom Phases: Quiescent



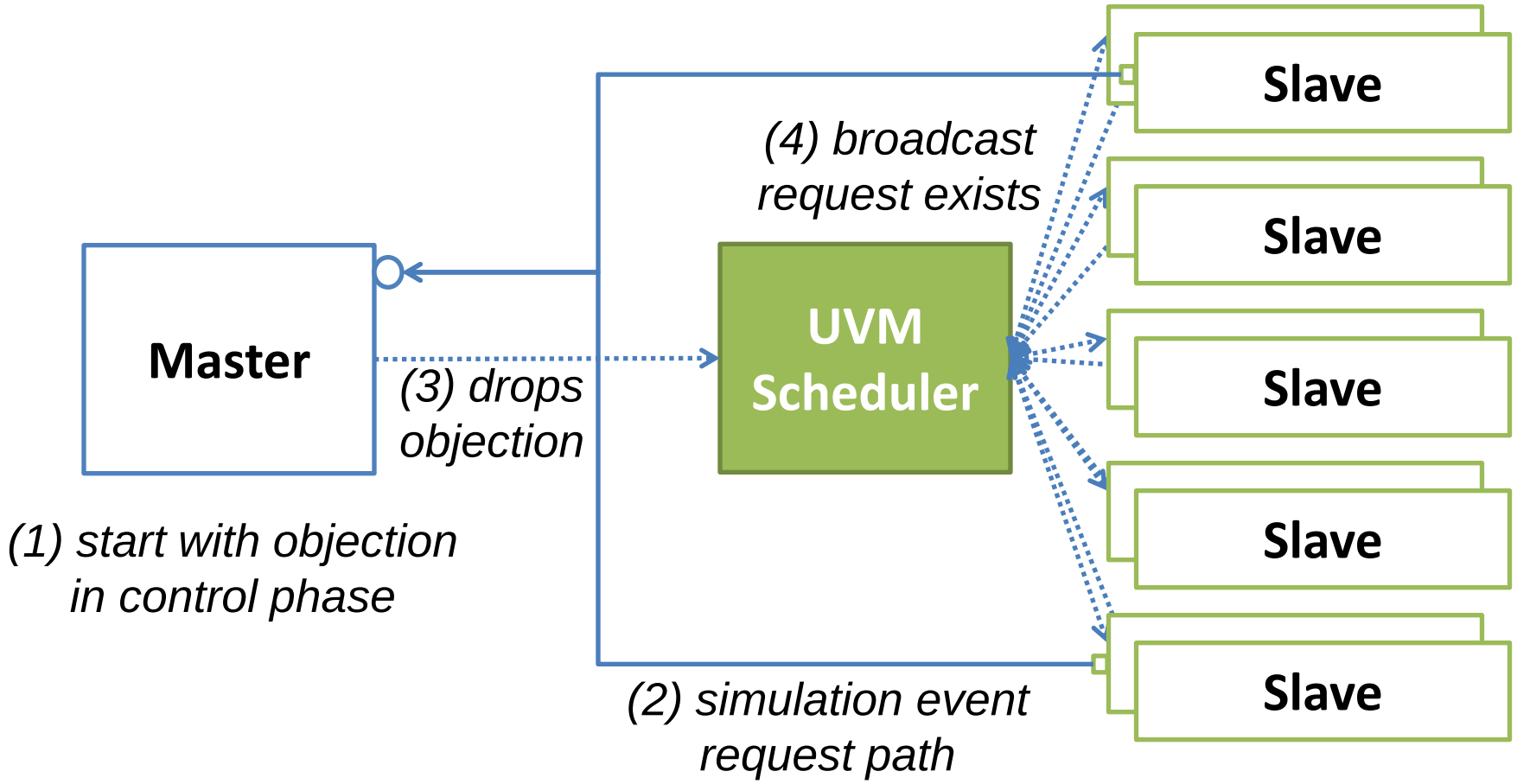
- Request to pause does not affect UVM main phase
- Loops through control phases

Custom Phases: Hard Reset



- Request to jumps back to UVM pre-reset phase
- Main phase is, effectively, cut-short

1-Master/n-Slaves Architecture



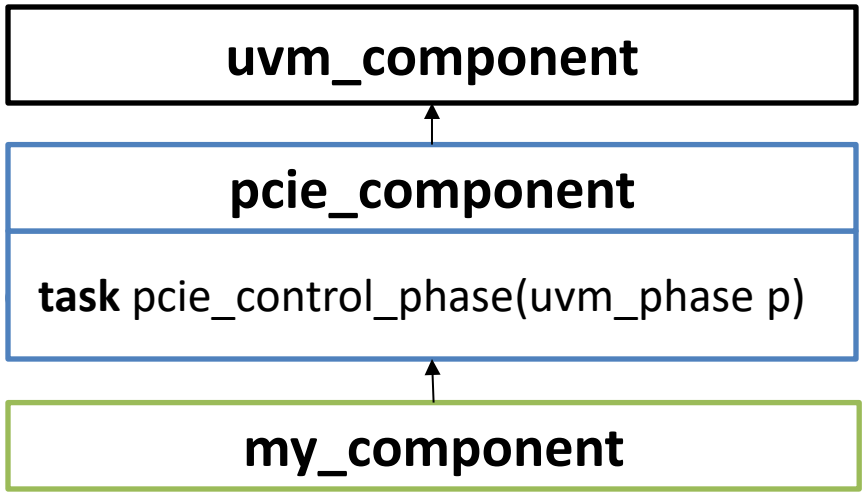
Agenda

- Why phasing
- Custom Phasing Architecture
- Custom Phasing Framework
- Master Component

Framework

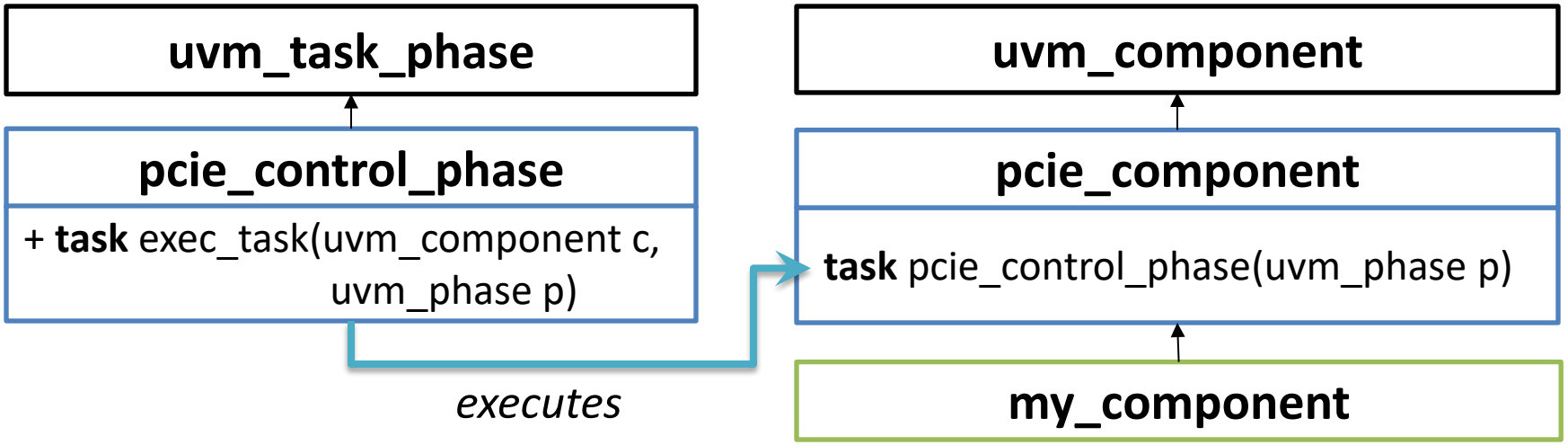
- Project-specific component base class library
- Custom phases and domain
- Phase proxy

Component Base Class Library



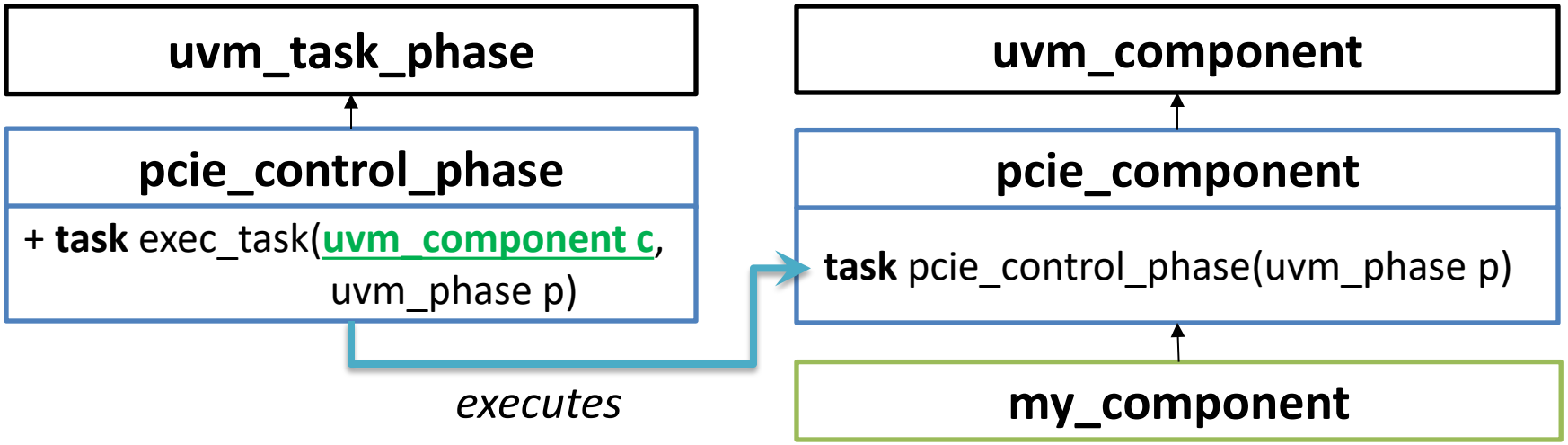
- Project-specific components inherit custom phases
  - Component, Environment, Agent, Driver, Monitor, etc.

Custom Phases



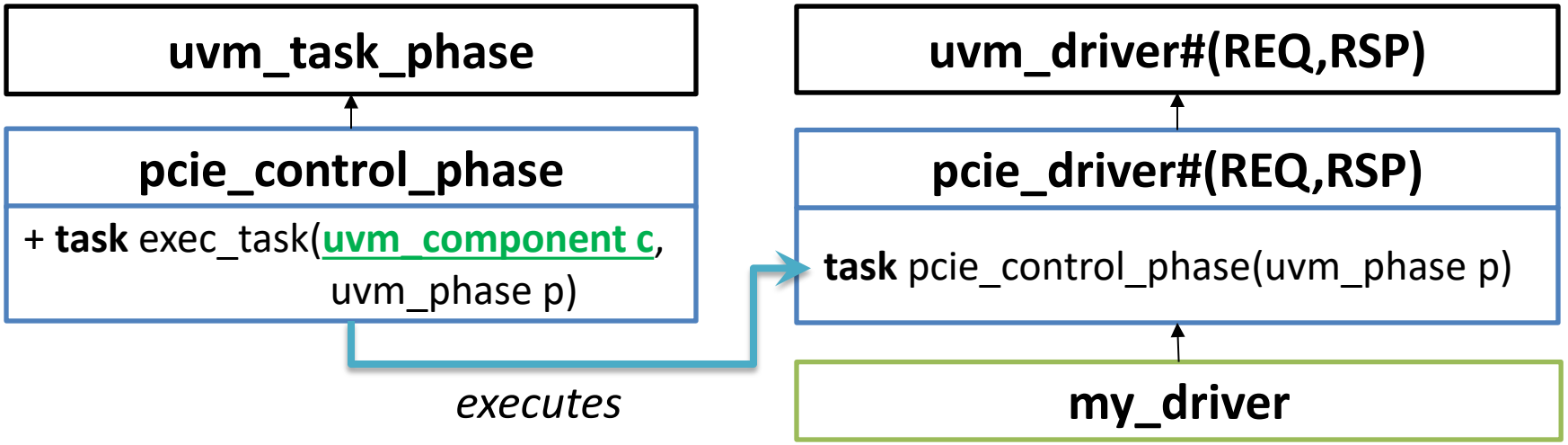
- UVM Scheduler calls exec_task in custom phase
- Custom phase calls task in base component classes

Custom Phases Issue #1: Parameterized Classes



- Custom phase is passed uvm_component reference
- Cast to project base component class

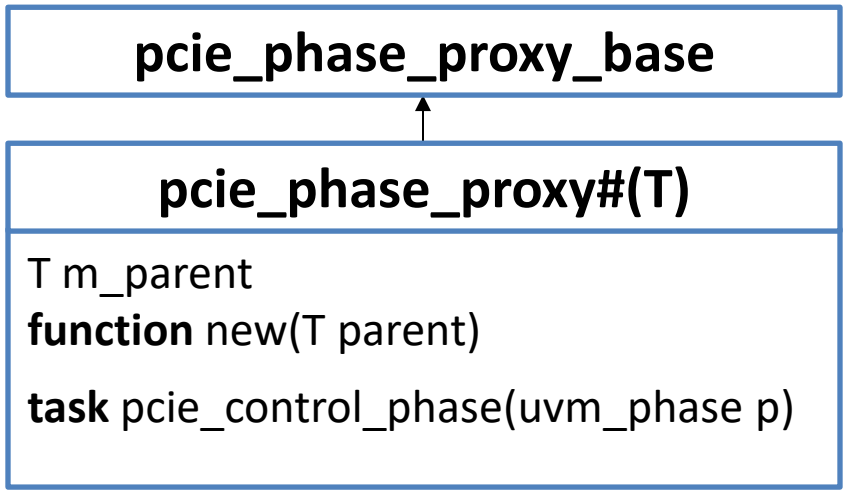
Custom Phases Issue #1: Parameterized Classes



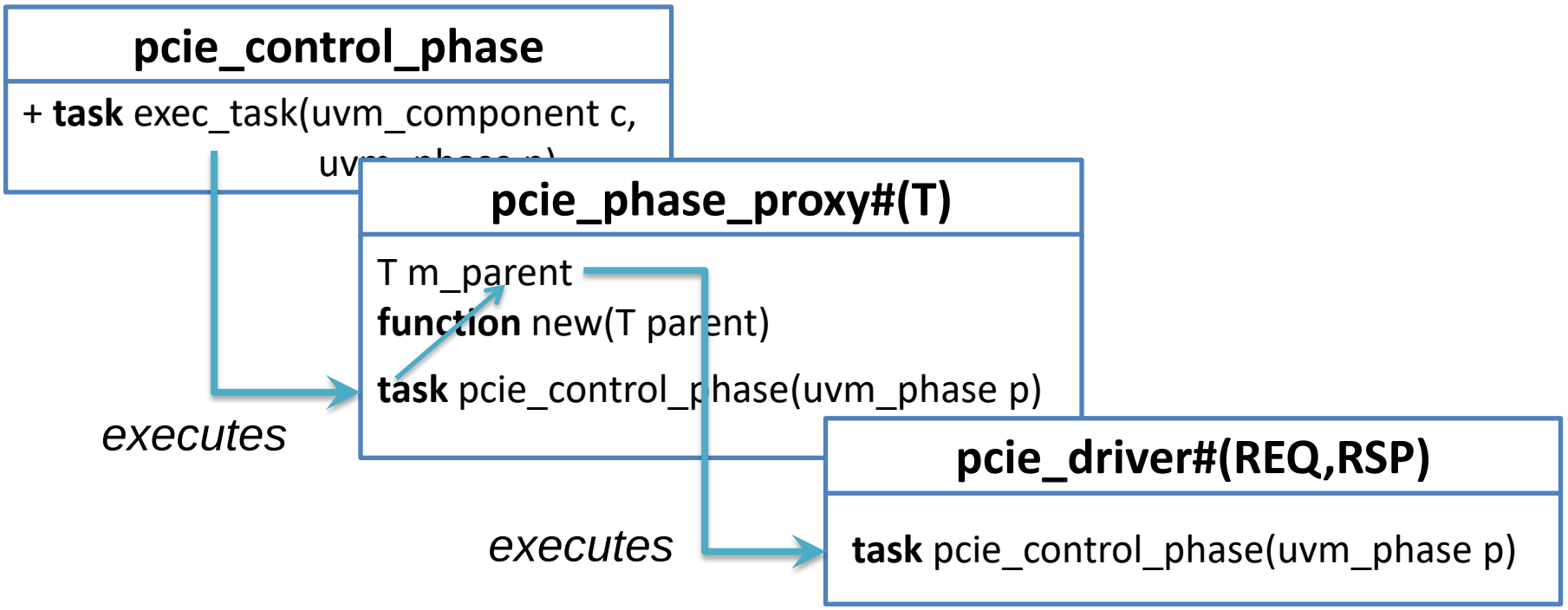
- No easy way to cast to parameterized class
  - Try all possible parameters?

Solution to #1: Phase Proxies

- Proxy contains typed reference to component
- Proxy base class
  - is not parameterized
  - has all (pure virtual) custom phase tasks
- Execute component task through proxy

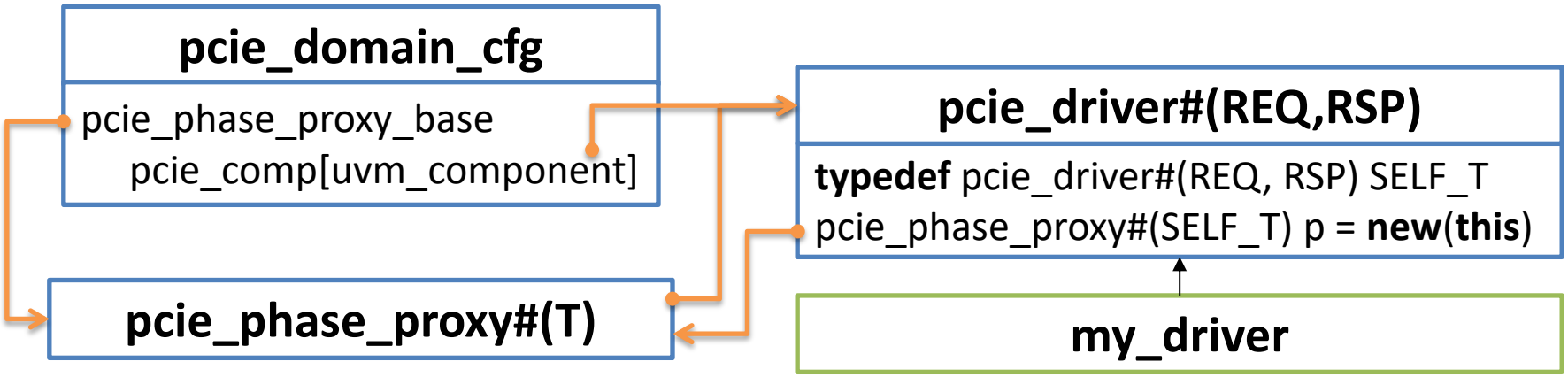


Solution to #1: Phase Proxies



- Phase executes task in proxy
- Proxy via parent reference executes task in driver

Phase Proxy Instantiation



- PCIe base component class
  - Instantiates proxy with reference to self
- Proxy registers self in global access associative array
  - Key is reference to base component class

Custom Phase Execution

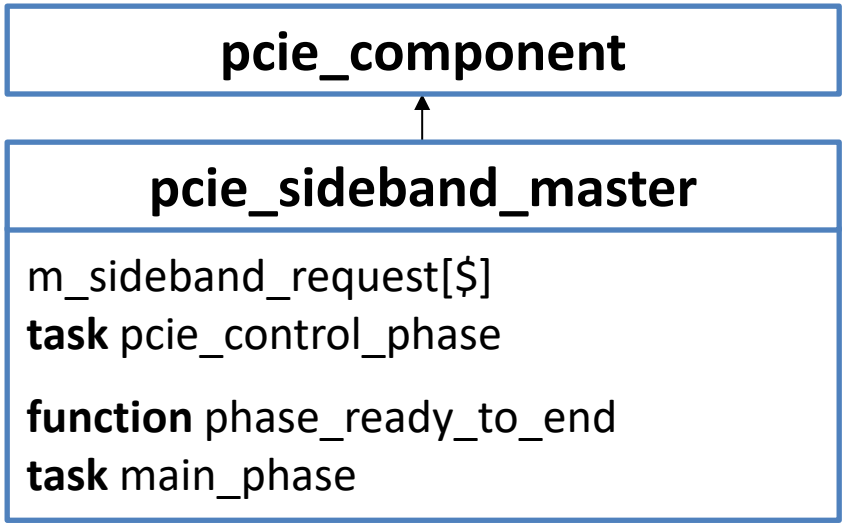
```
class pcie_control_phase extends uvm_task_phase;
  task exec_task(uvm_component c, uvm_phase ph);
    pcie_phase_proxy_base p =
        pcie_domain_cfg::is_pcie_comp(c);
    if (p != null)
      p.pcie_control_phase(ph);
  endtask
endclass
```

- Execution is straightforward with phase framework
  - All custom phases are implemented the same

Agenda

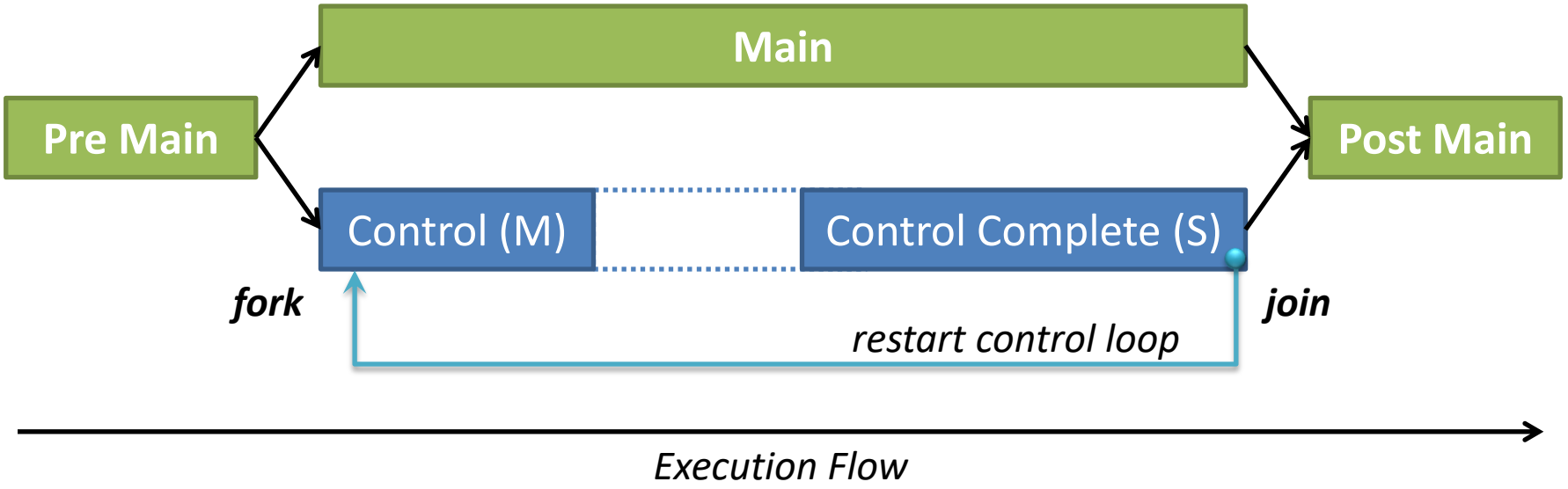
- Why phasing
- Custom Phasing Architecture
- Custom Phasing Framework
- Master Component

Master Component



- Is a pcie_component
- Processes requests
  - pcie_control_phase
- Processes phase jumps
  - phase_ready_to_end
- Monitor main phase for phase exit

Custom Phasing Issue #2: Main Phase Exit



- When main phase objections are all dropped, scheduler synchronizes main with child and sibling phases before phase_ready_to_end is executed

Solution to #2: Monitor Objections in Main Phase

- Master component implements main phase
  - Wait for all objections dropped up to uvm_root
  - Disables PCIe side-band control loop
    - No more phases execute allowing all phases to exit
  - Requests control phase to drop objection and exit

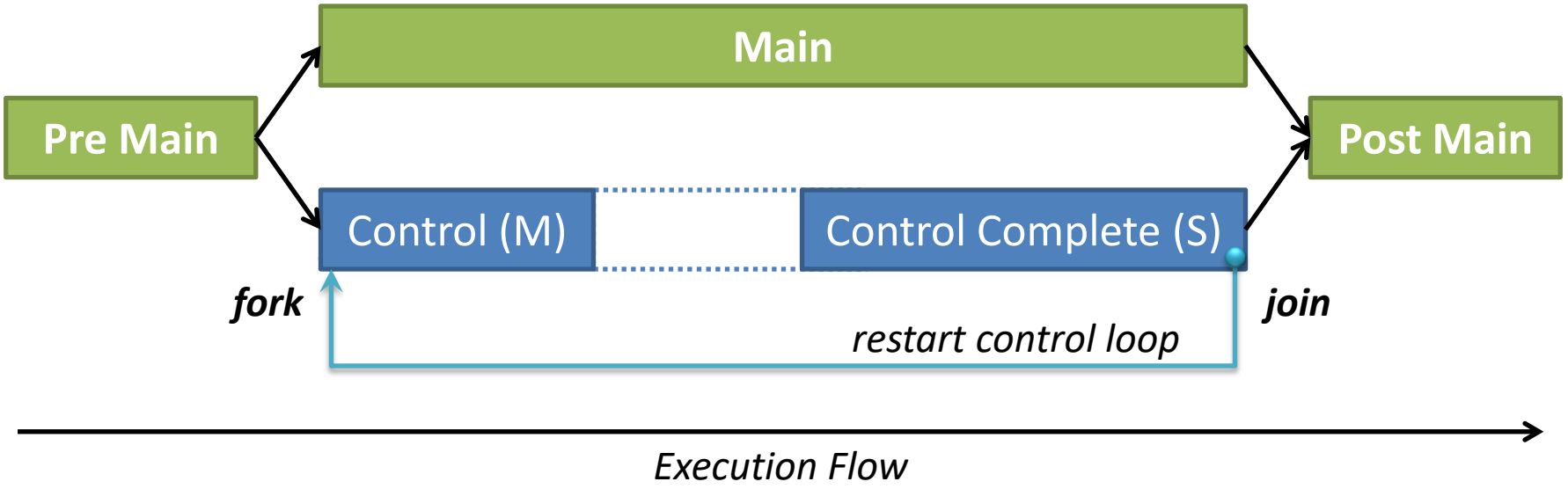
- Control phase sees exit request and drops objection

Solution to #2: Monitor Objections in Main Phase

```
class pcie_sideband_master extends pcie_component;
  task main_phase(uvm_phase ph);
    uvm_root top = uvm_root::get();
    uvm_objection done = ph.get_objection();
    if(!done.m_top_all_dropped)
      done.wait_for(UVM_ALL_DROPPED, top);
    pcie_domain_cfg::disable_sideband_phases();
    request_exit_control_phase();
  endtask
endclass
```

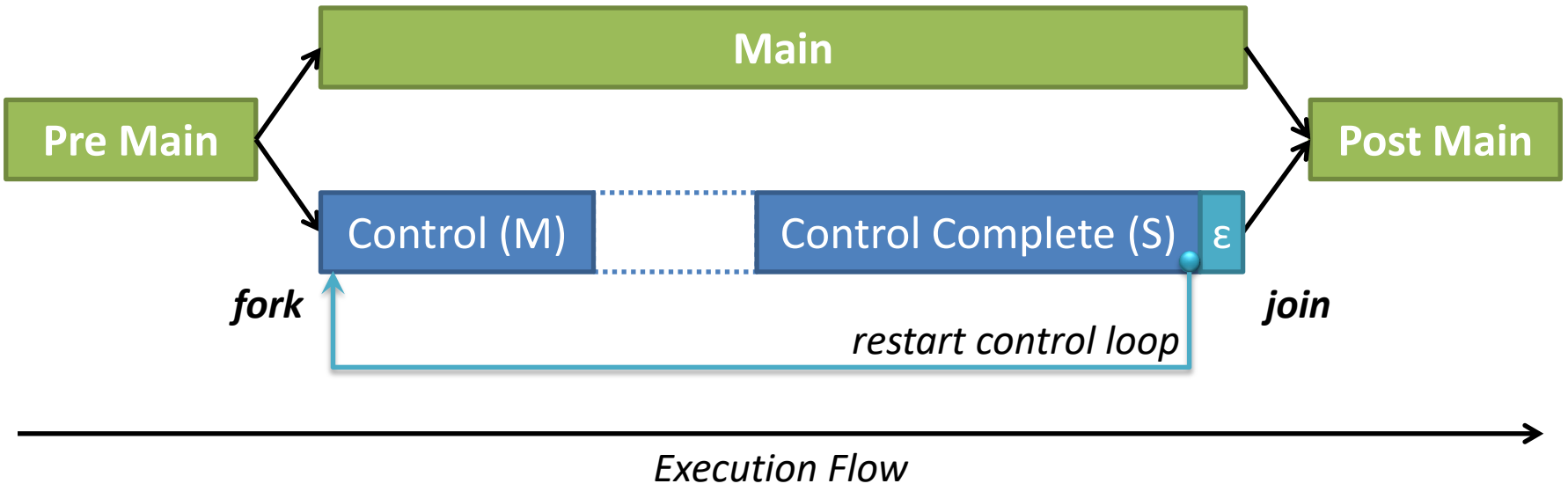
- Use reference uvm_root in wait_for all dropped

Custom Phasing Issue #3: Control Complete Exit



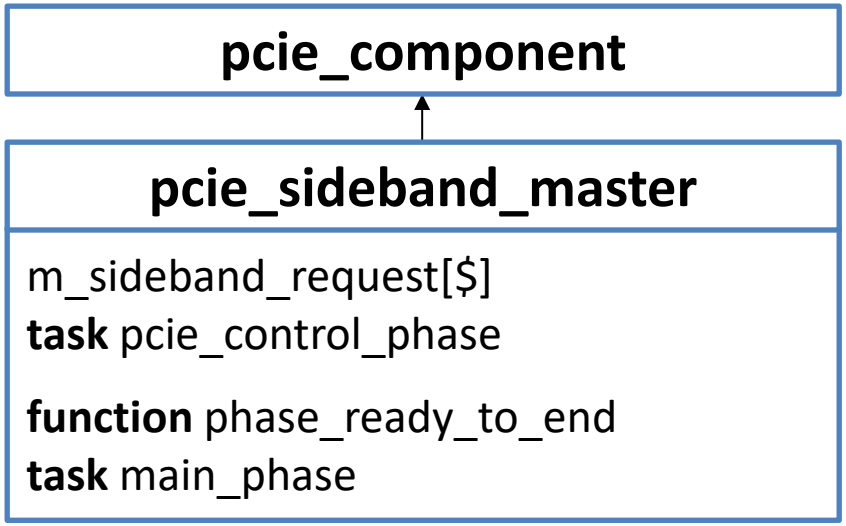
- Control complete has same as issue #2
  - Jump to control at phase_ready_to_end(control-complete)

Solution to #3: Terminal Phase



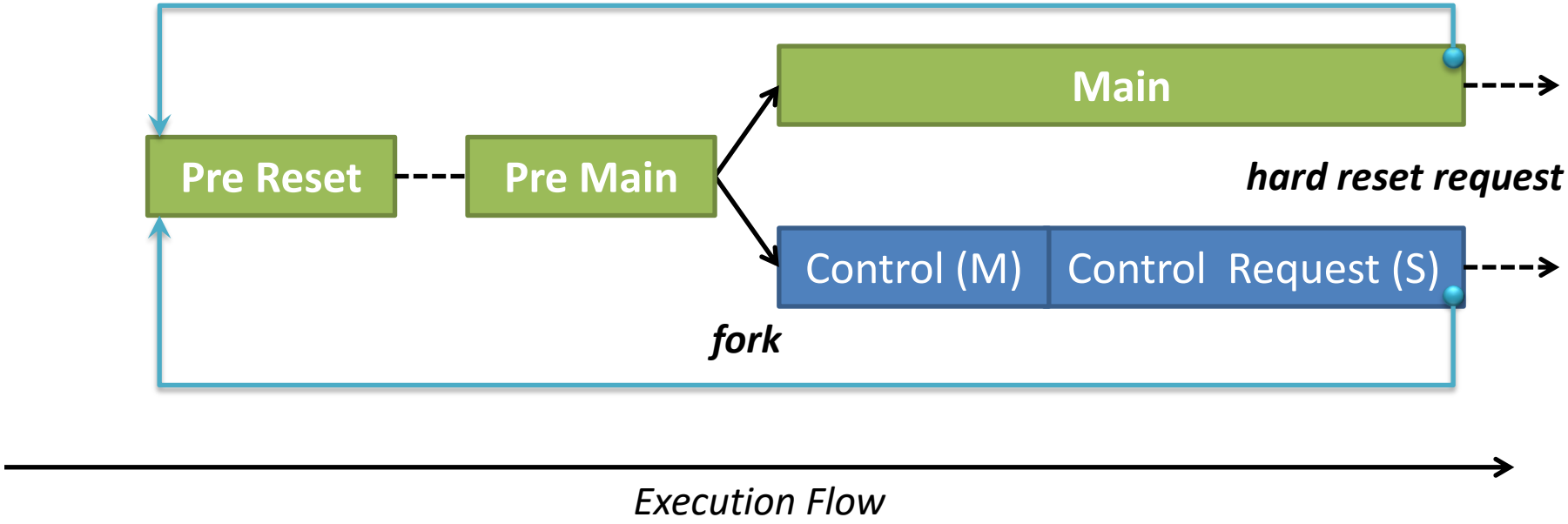
- Dummy terminal phase synchronizes with main
- Control complete executes phase_ready_to_end

Master Component



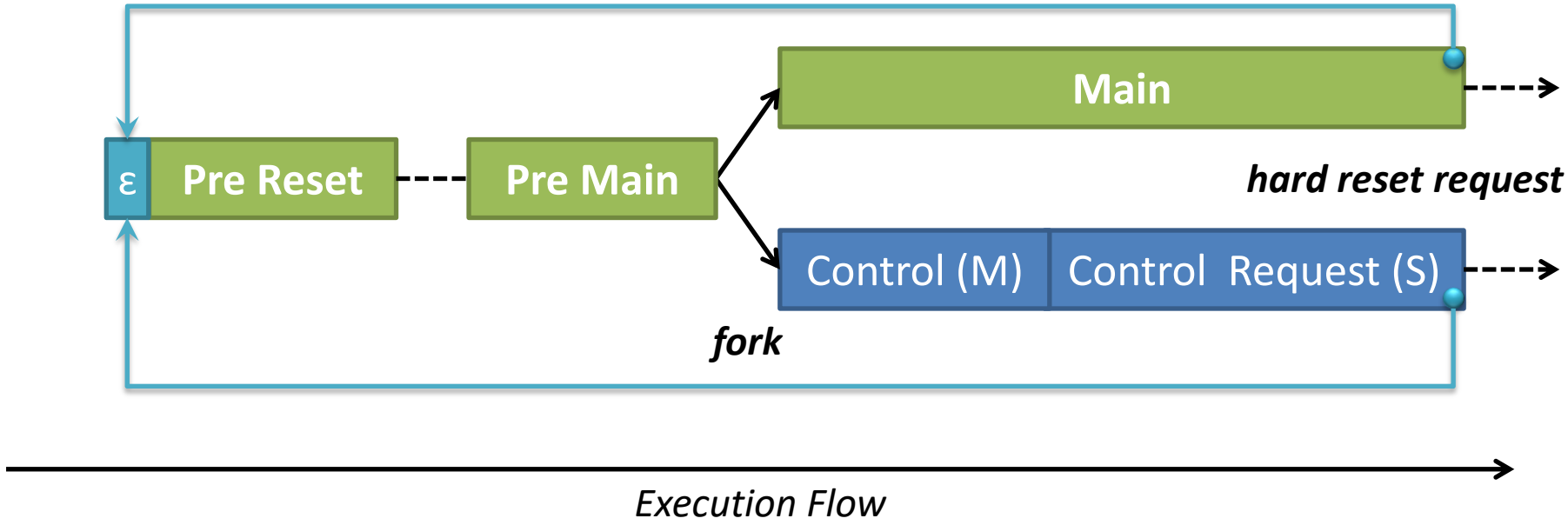
- Is a pcie_component
- Processes requests
  - pcie_control_phase
- Processes phase jumps
  - phase_ready_to_end
- Monitor main phase for phase exit

Custom Phasing Issue #4: Multiple Phase Executions



- Simultaneous jumps and executes target phase
  - Threads merge at end of target phase

Solution to #4: Dummy Target Phase



- Dummy phase executes no task
- Dummy phase allows threads to join

Master Phase Jumping

```
class pcie_sideband_master extends pcie_component;
  function phase_ready_to_end(uvm_phase ph);
    uvm_phase imp = ph.get_imp();
    if (imp == pcie_control_complete_phase::get())
     ph.jump(pcie_control_phase::get());
    if (imp == pcie_control_request_phase::get() &&
        request == HARD_RESET)
pcie_domain::jump_all(pcie_dummy_phase::get());
  endfunction
endclass
```

- Local jump with phase reference
- Global jump with domain reference

Results

- Implemented two classes of simulation events:
  - Stay within main phase
  - Jump out of main phase
- Testing implementation was straightforward
  - Sequencers halted traffic by asserting a class flag
    - Sequences checked flag before generating next stimulus
  - Sequencers killed traffic if jump out of main was expected

Conclusions

- Why phasing
- Custom Phasing Architecture
- Custom Phasing Framework
  - Parameterized component classes: use phase proxy
- Master Component
  - Ending main phase: monitor objections
  - Ending control-complete phase: dummy terminal phase
  - Multiple threads in jump_all: dummy target phase

Questions

Thank You!

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA