Global Event Handling with UVM Custom Phasing

Jeremy Ridgeway, Dolly Mehta

Avago Technologies, Ltd.

September 18, 2015

SNUG AUSTIN

Agenda

Problem

Why Phasing

Custom Phasing Architecture

Custom Phasing Framework

Master Component

Conclusions

Problem

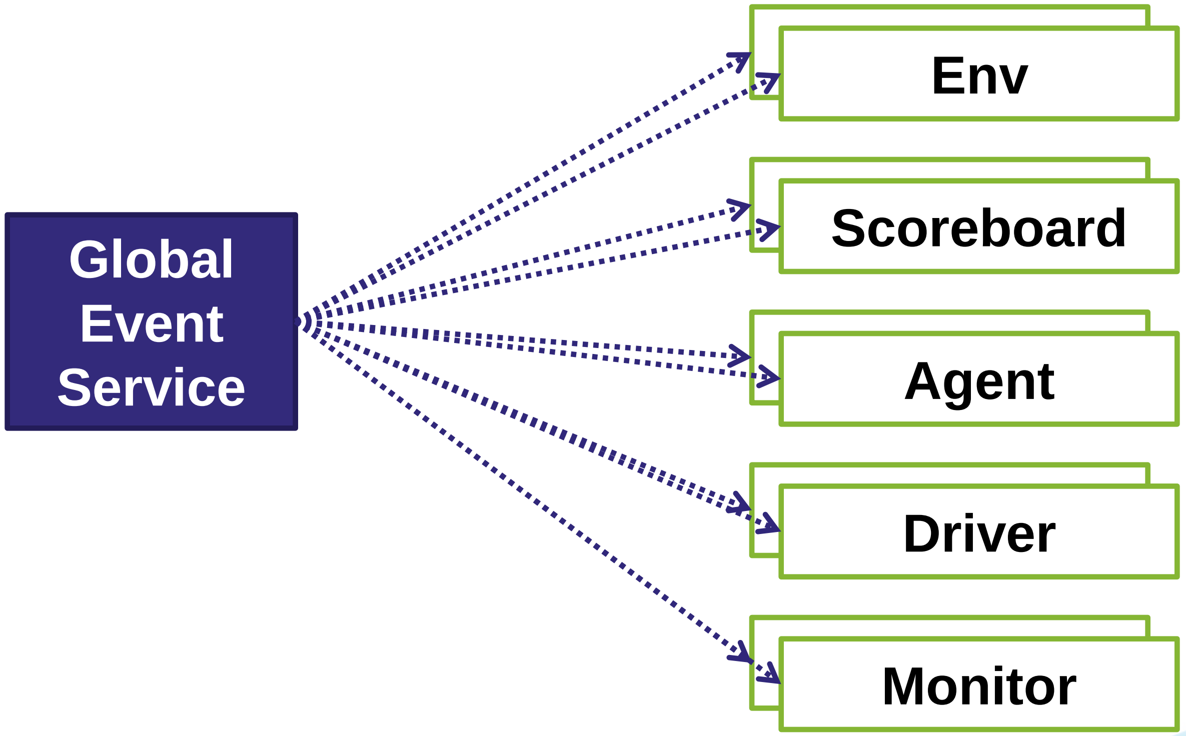
- Must perform **catastrophic** event mid-simulation
  - HARD RESET

- Find a common way to handle global events
  - Quiescence

- How to prepare the environment *before* the event?

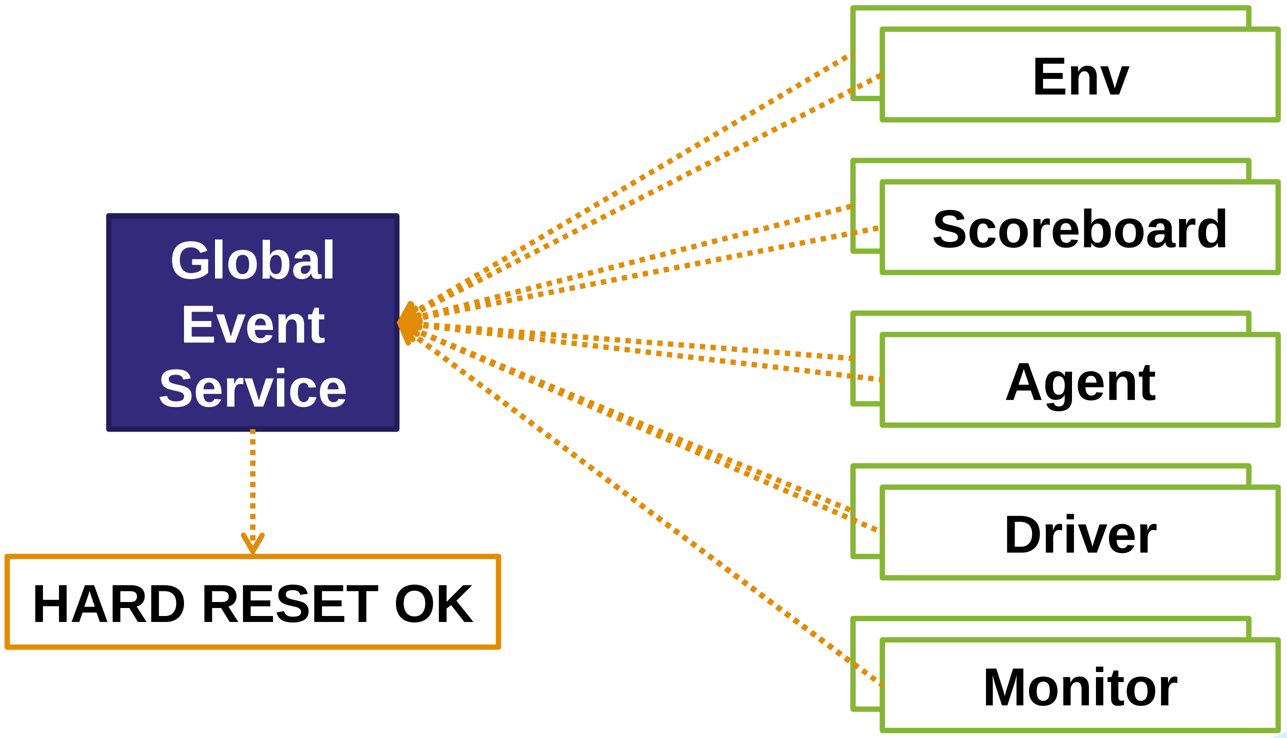
Goals

- Centralized event notification service
- Push style notification



Goals

- Centralized event notification service
- Push style notification
- Handshake to ensure environment is ready



Events to Handle

- Catastrophic
  - Hard reset
  - DUT Configuration change

- Non-catastrophic
  - Quiescence

Solution: Phasing!

- Already connected with all components
- Extensible for custom phases

- Centralized control
  - Provides push-style notification via tasks
  - Passive reception
  - Easy to wait for ready

- <u>Nothing new to learn</u>

Agenda

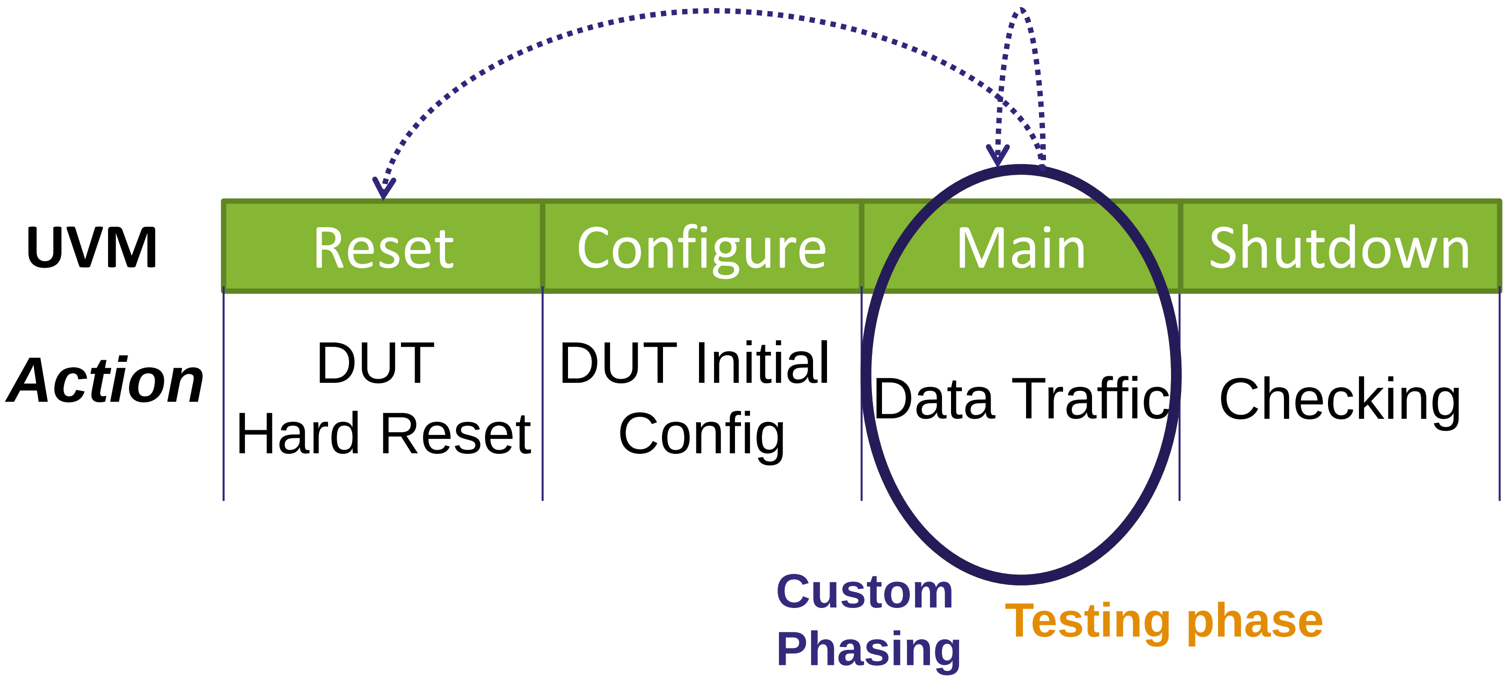
Overall Run-time Phase Schedule

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

Testing phase

- Not changing the overall phasing schedule
- Add a parallel schedule

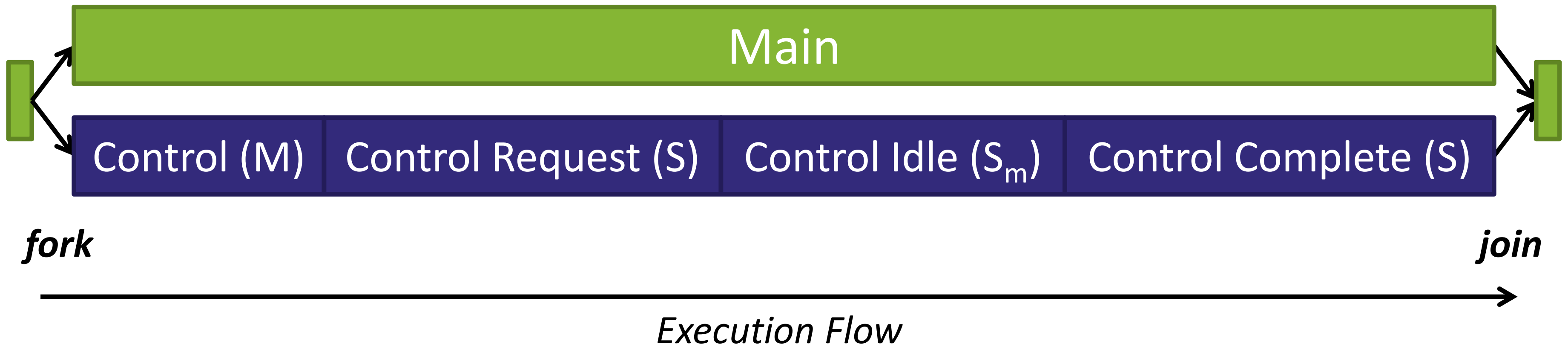
Overall Run-time Phase Schedule



- Hard reset jumps from Main to Reset Phase
- Quiescence in Main Phase only
- Custom phasing in Main Phase

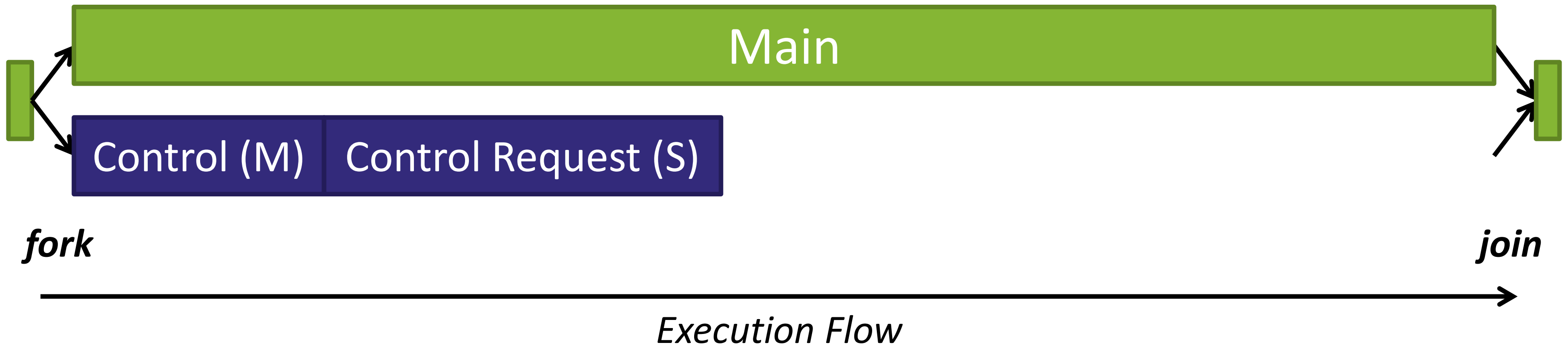
Custom Phase Schedule

Parallel with Main Phase Only



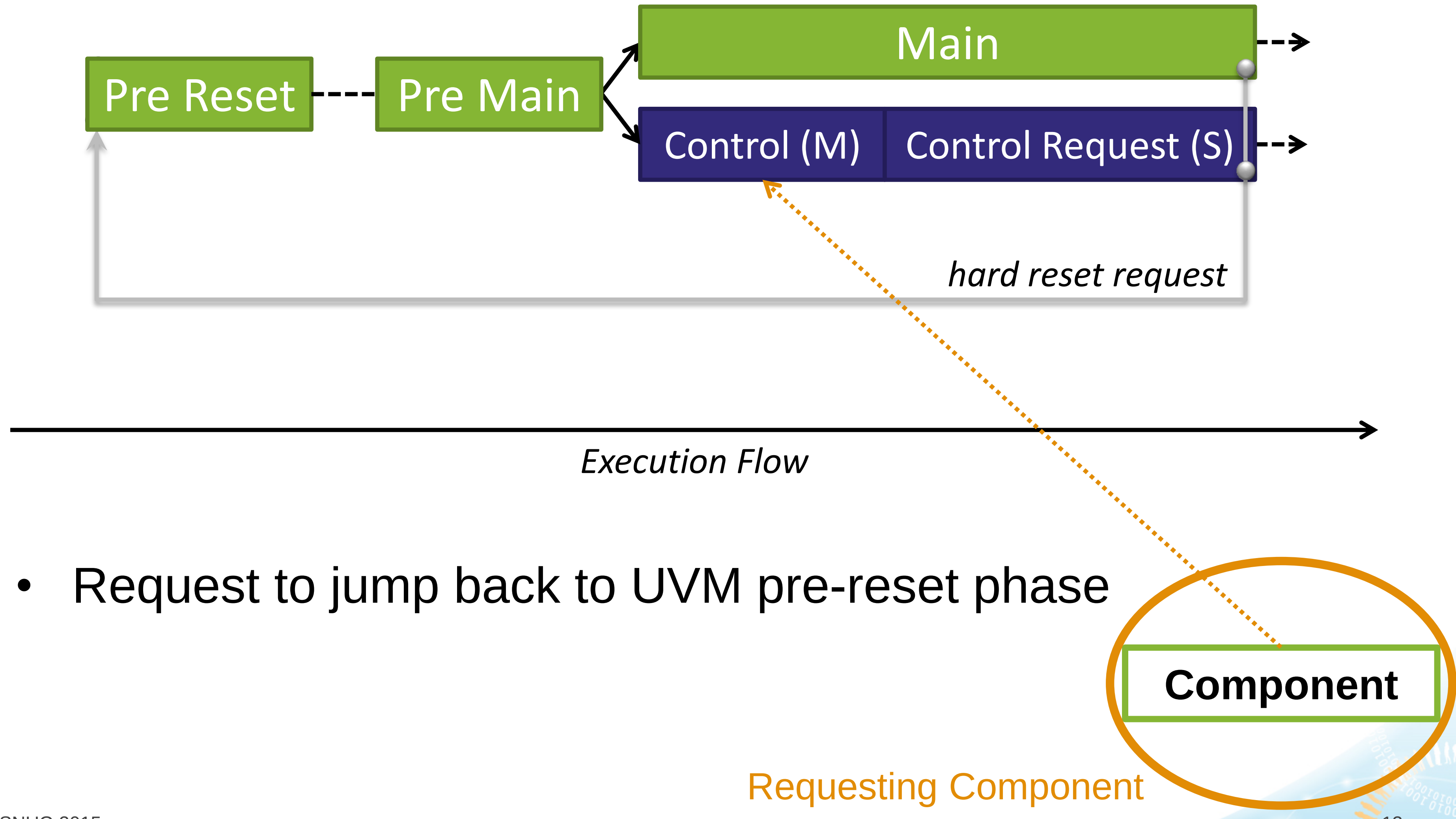
Custom Phase Schedule

Parallel with Main Phase only



- Control – implemented in 1 component only (Master)
- Control Request
  - Push notification: prepare for event (raise objection)
  - Handshake: I am prepared (lower objection)

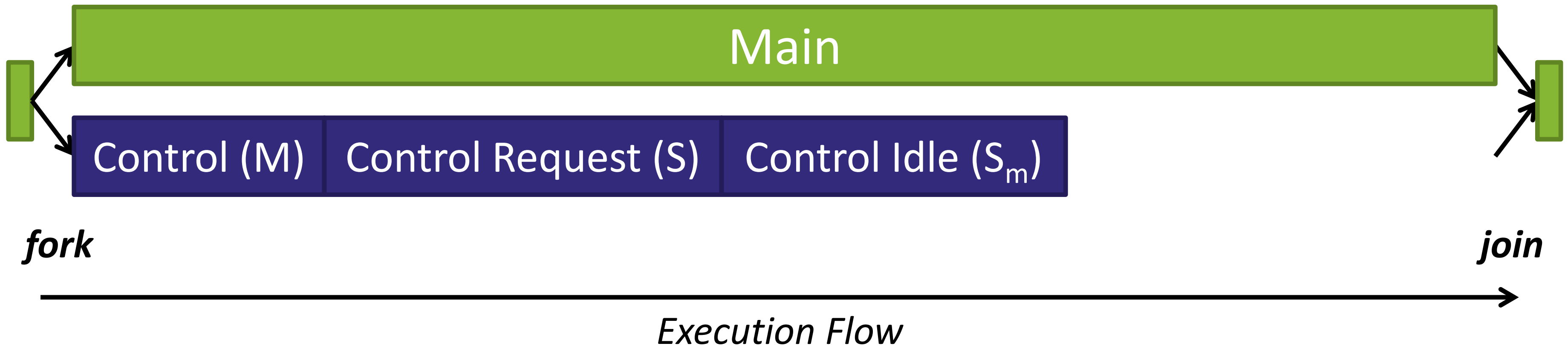
Handling Hard Reset



- Request to jump back to UVM pre-reset phase

Custom Phase Schedule

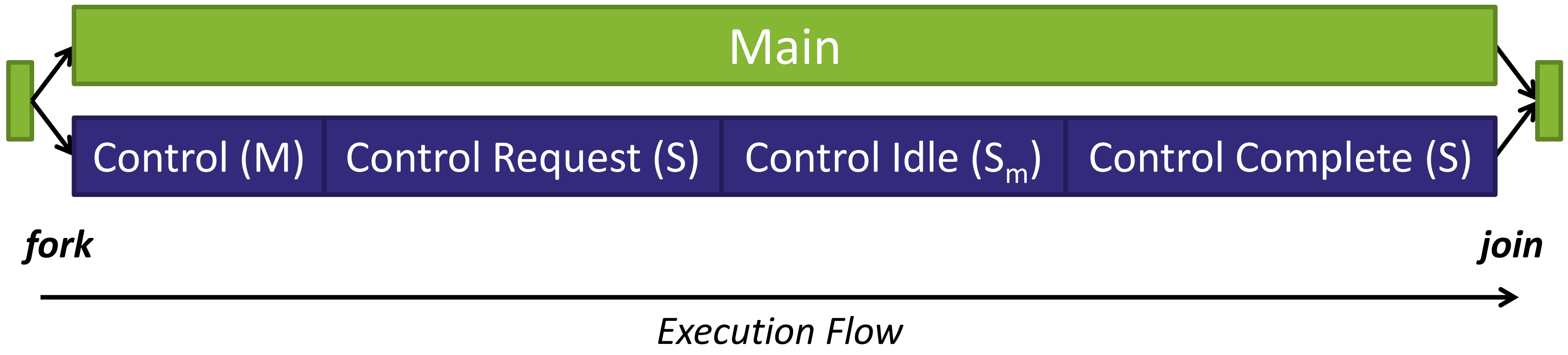
Parallel with Main Phase only



- Control – implemented in 1 component only (Master)
- Control Request
  - Push notification: prepare for event (raise objection)
  - Handshake: I am prepared (lower objection)
- Control Idle – Quiescence

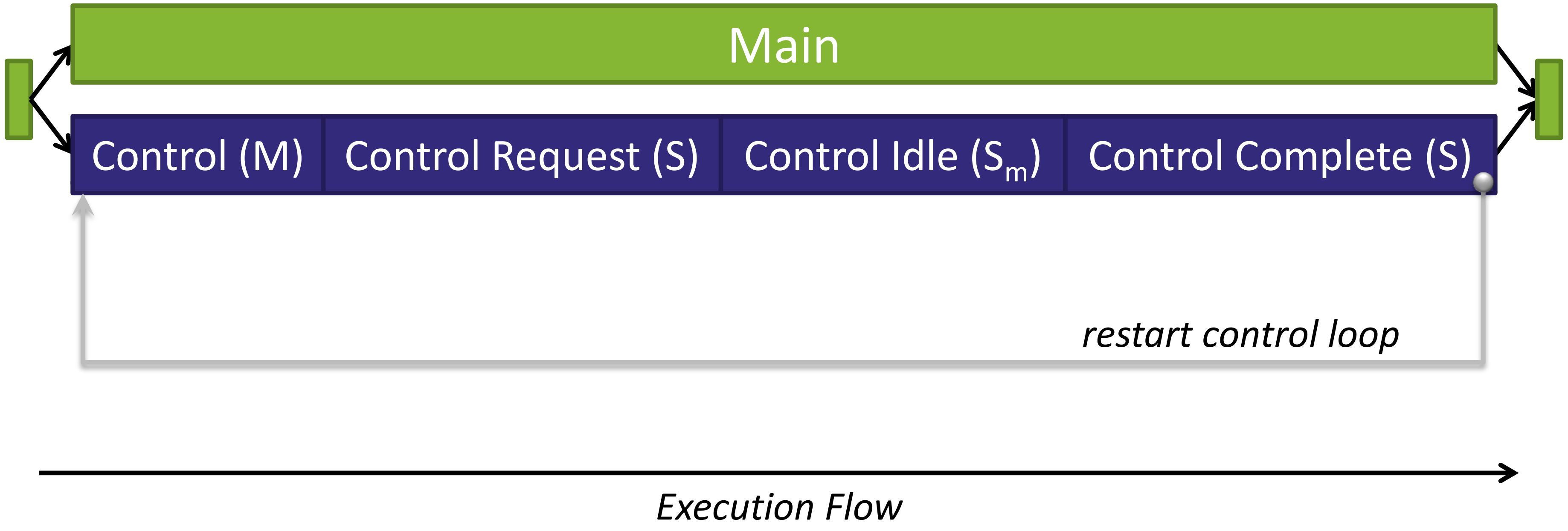
Custom Phase Schedule

Parallel with Main Phase only



- Control – implemented in 1 component only (Master)
- Control Request
  - Push notification: prepare for event (raise objection)
  - Handshake: I am prepared (lower objection)
- Control Idle – Quiescence
- Control Complete – Quiescence complete

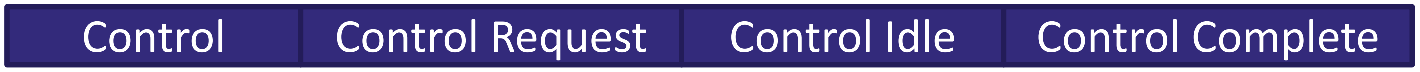
Restart after Quiescence



- Quiescence does not affect UVM main phase
- Loop back and restart control phases

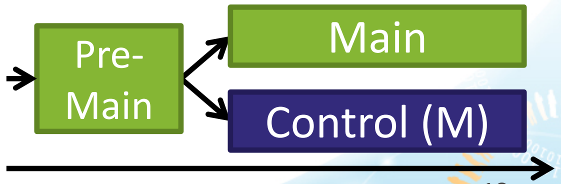
Agenda

Custom Phasing Framework

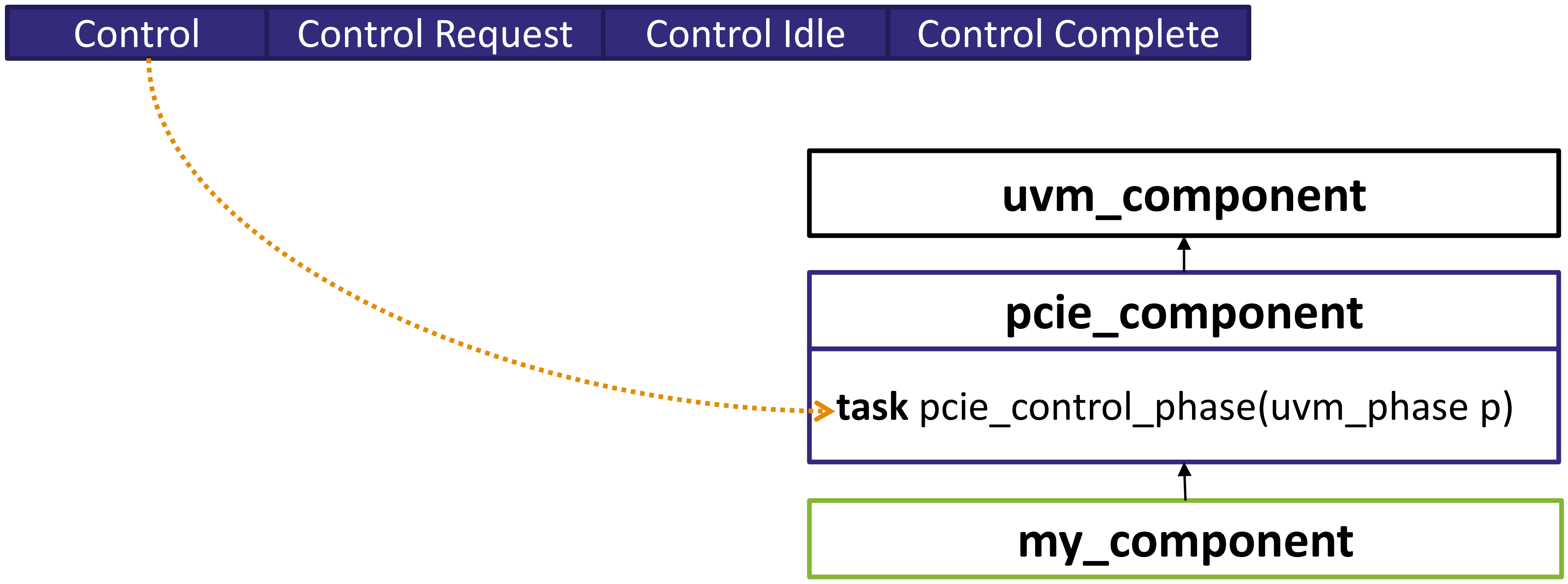


- Define run-time phase schedule
- Identify where phases will execute

- Project-specific component base classes
- Schedule custom phases and their domain
- Phase proxy

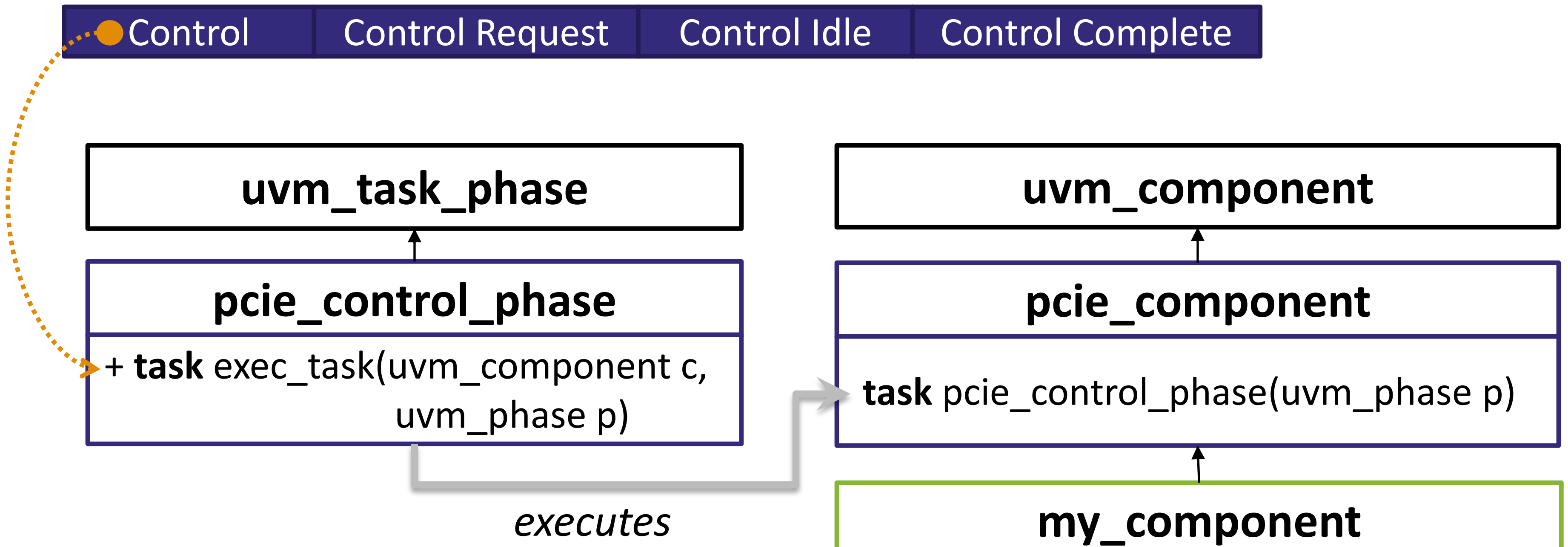


Component Base Classes



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

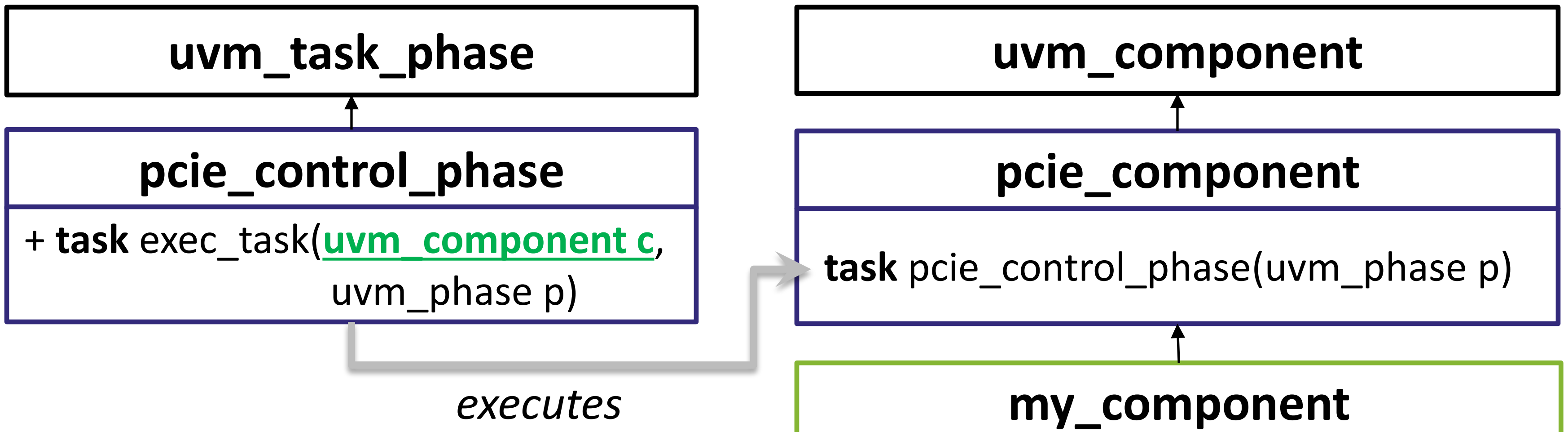
Phases are Classes



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

Custom Phases Issue #1

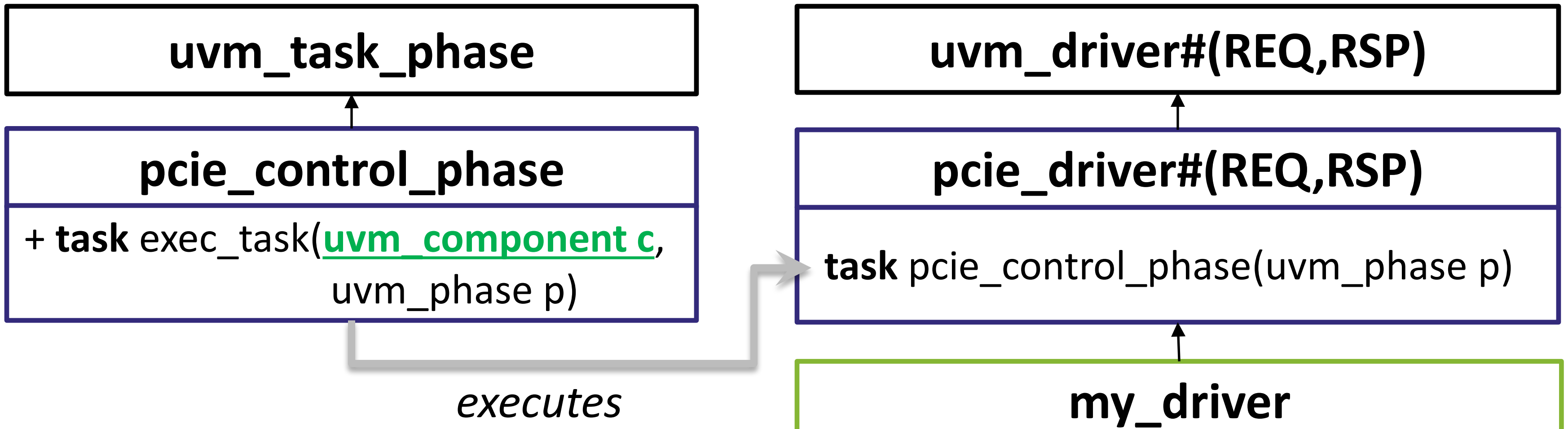
Parameterized Classes are Hard to Cast



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

Custom Phases Issue #1

Parameterized Classes are Hard to Cast



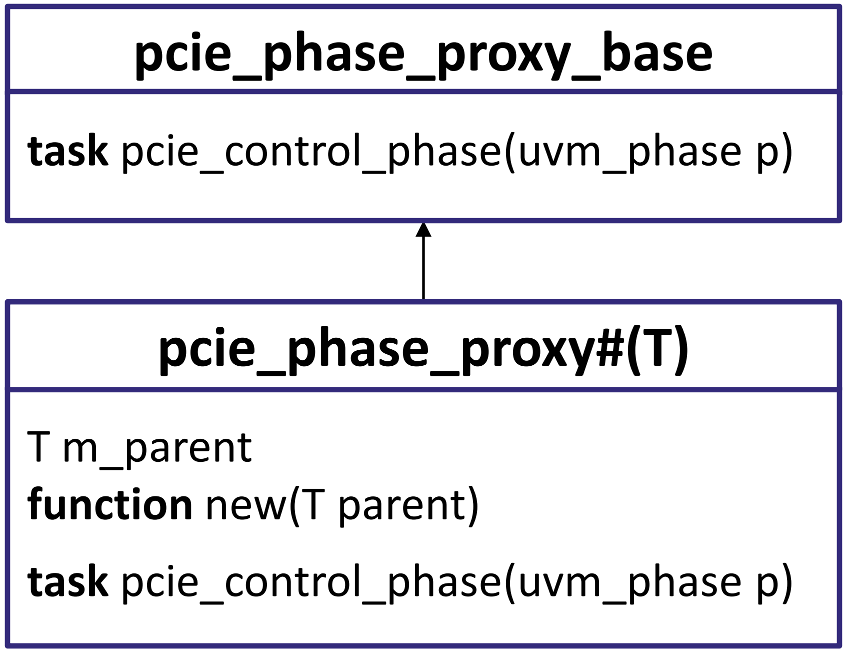
uvm_task_phase
pcie_control_phase
+ **task** exec_task(uvm_component c, uvm_phase p)
executes
uvm_driver#(REQ,RSP)
pcie_driver#(REQ,RSP)
task pcie_control_phase(uvm_phase p)
my_driver

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

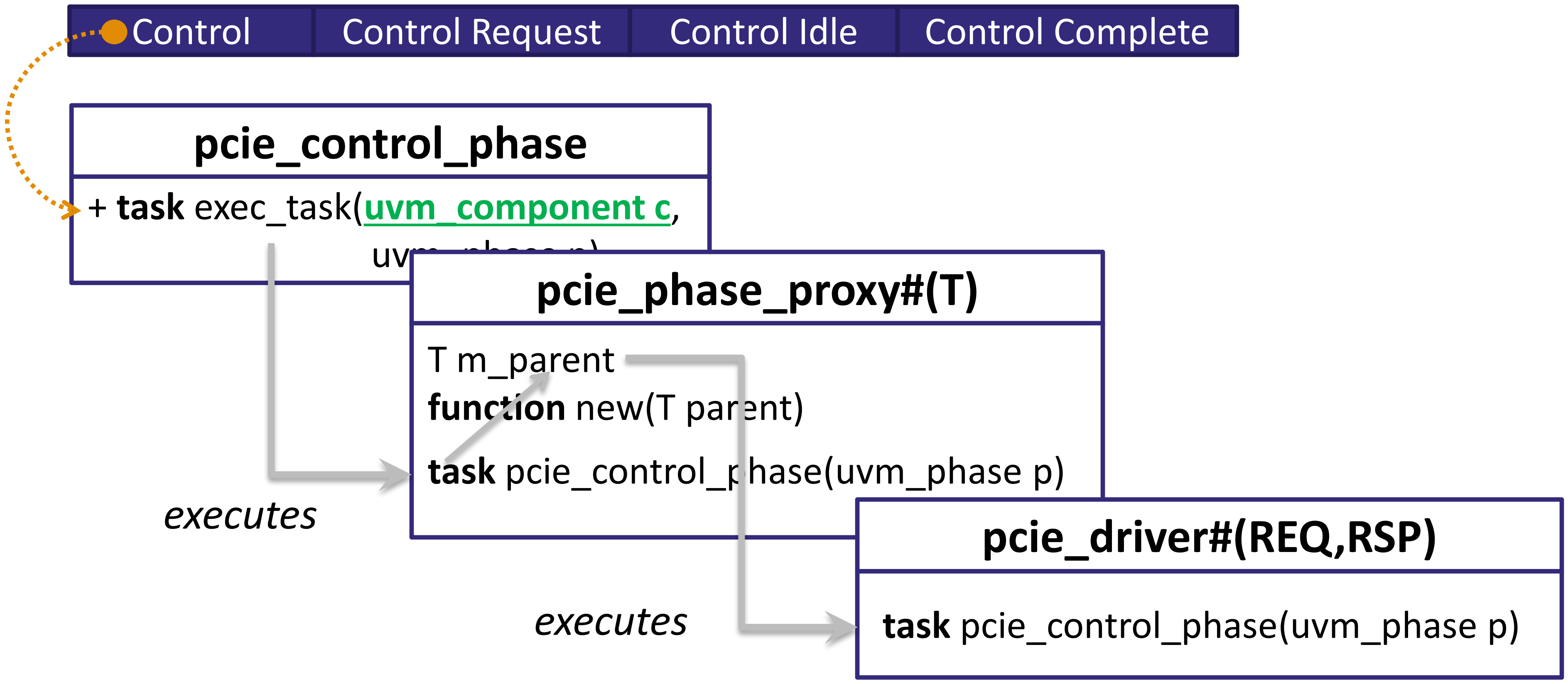
Phase Proxy Class

Solution to issue #1

- Similar to UVM Factory
- Instantiated inside project-specific component
- Execute component custom phase task through proxy



Using a Phase Proxy



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

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

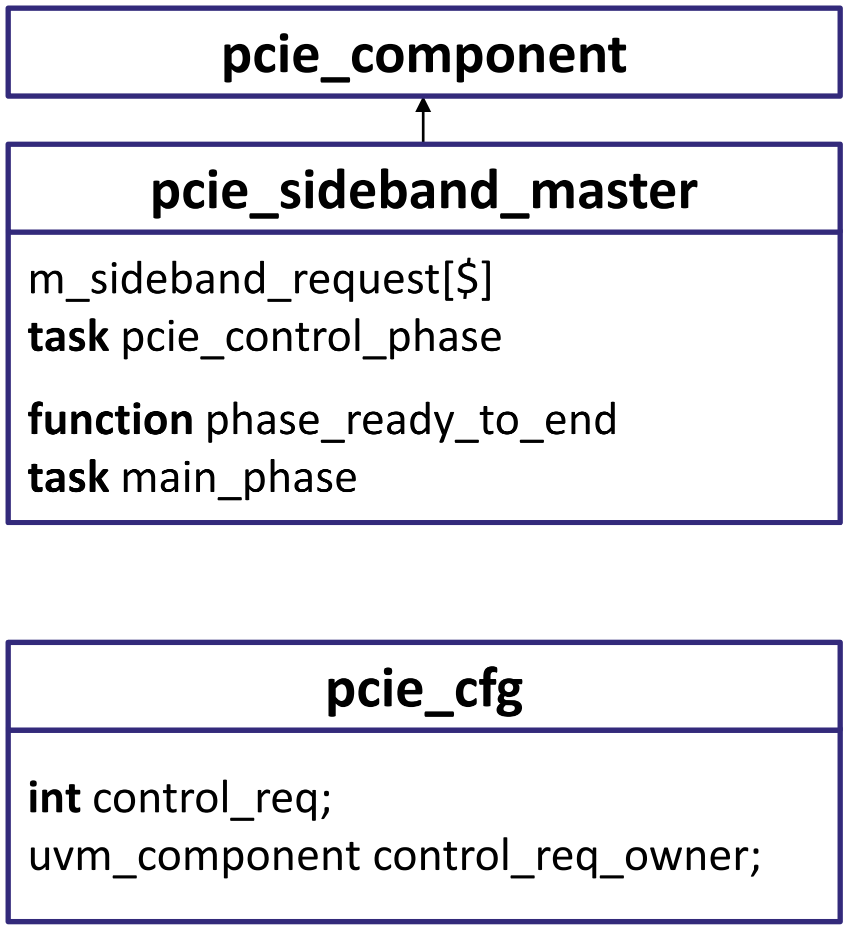
Custom Phase Execution

```
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

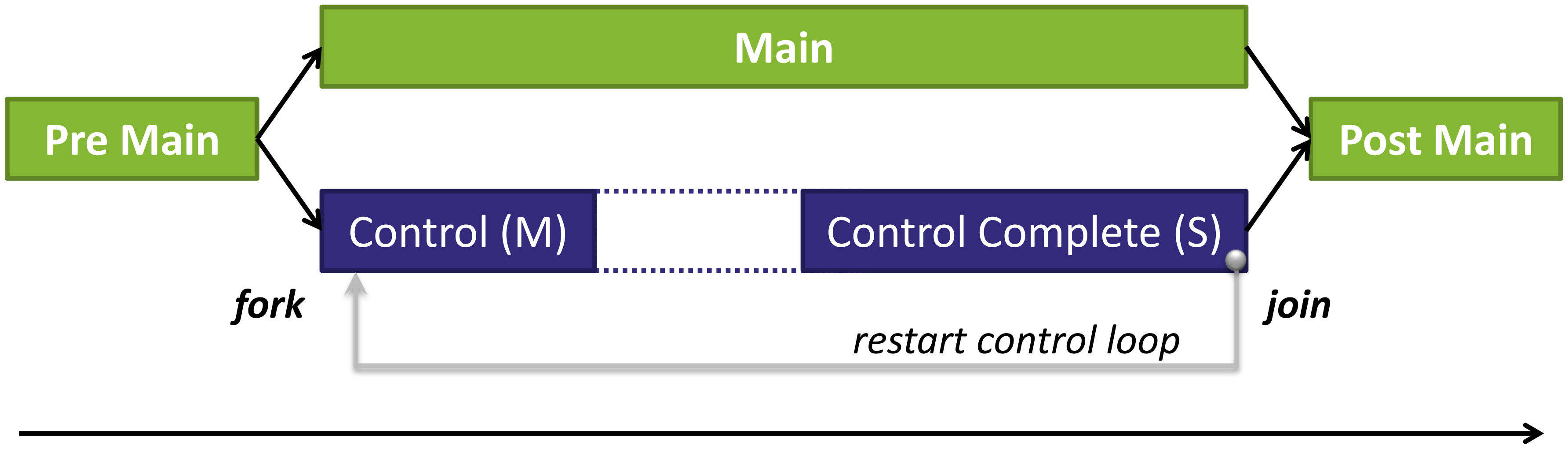
Master Component



- Is a pcie_component
- Processes requests
  - pcie_control_phase
- Posts to bulletin board
- Processes phase jumps
  - phase_ready_to_end
- Monitor main phase for phase exit

Custom Phasing Issue #2

Main Phase Exit



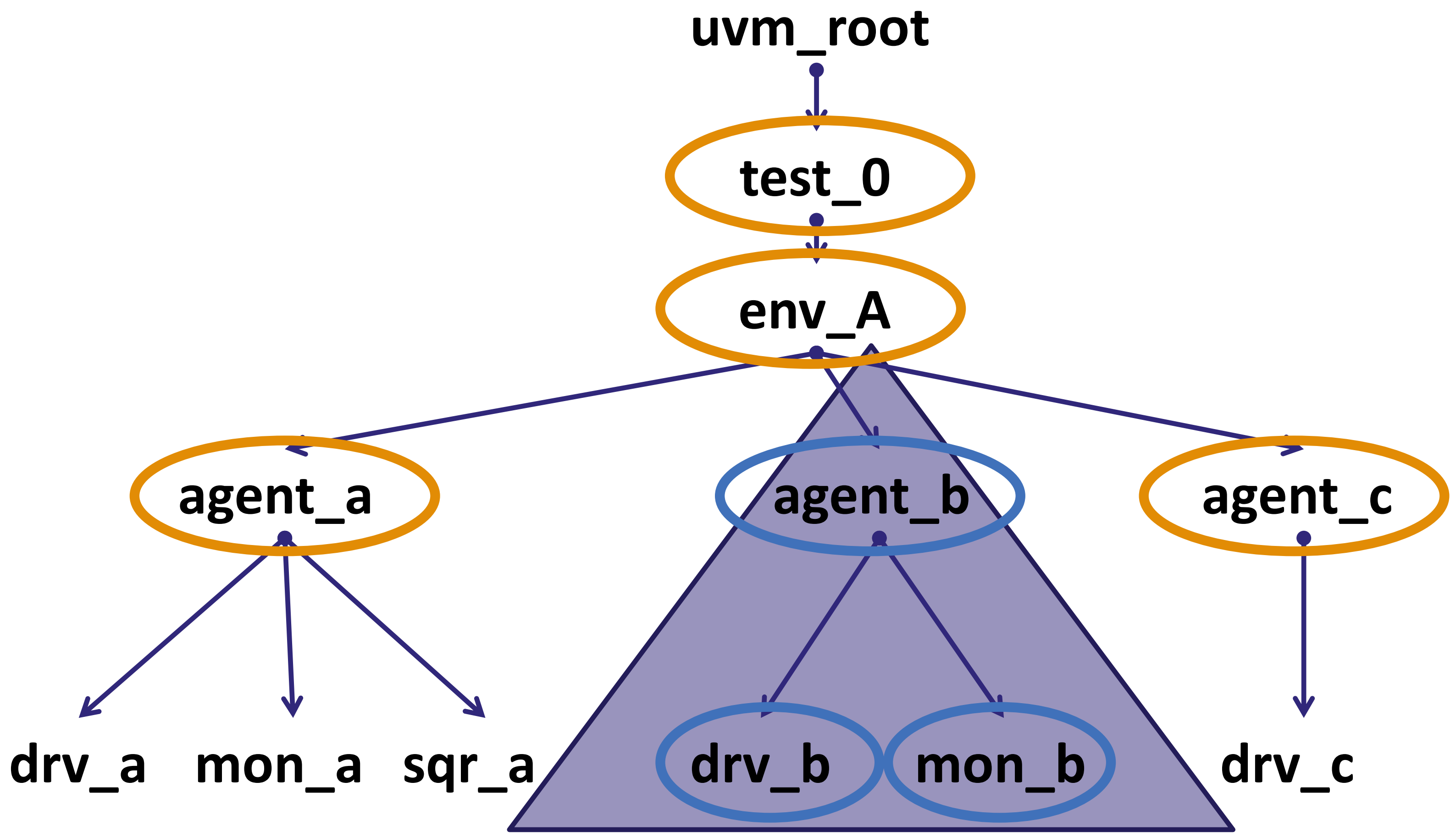
- When main phase objections are all dropped, scheduler synchronizes main with child and sibling phases before phase_ready_to_end is executed

Solution to issue #2

Monitor Objections in Main Phase

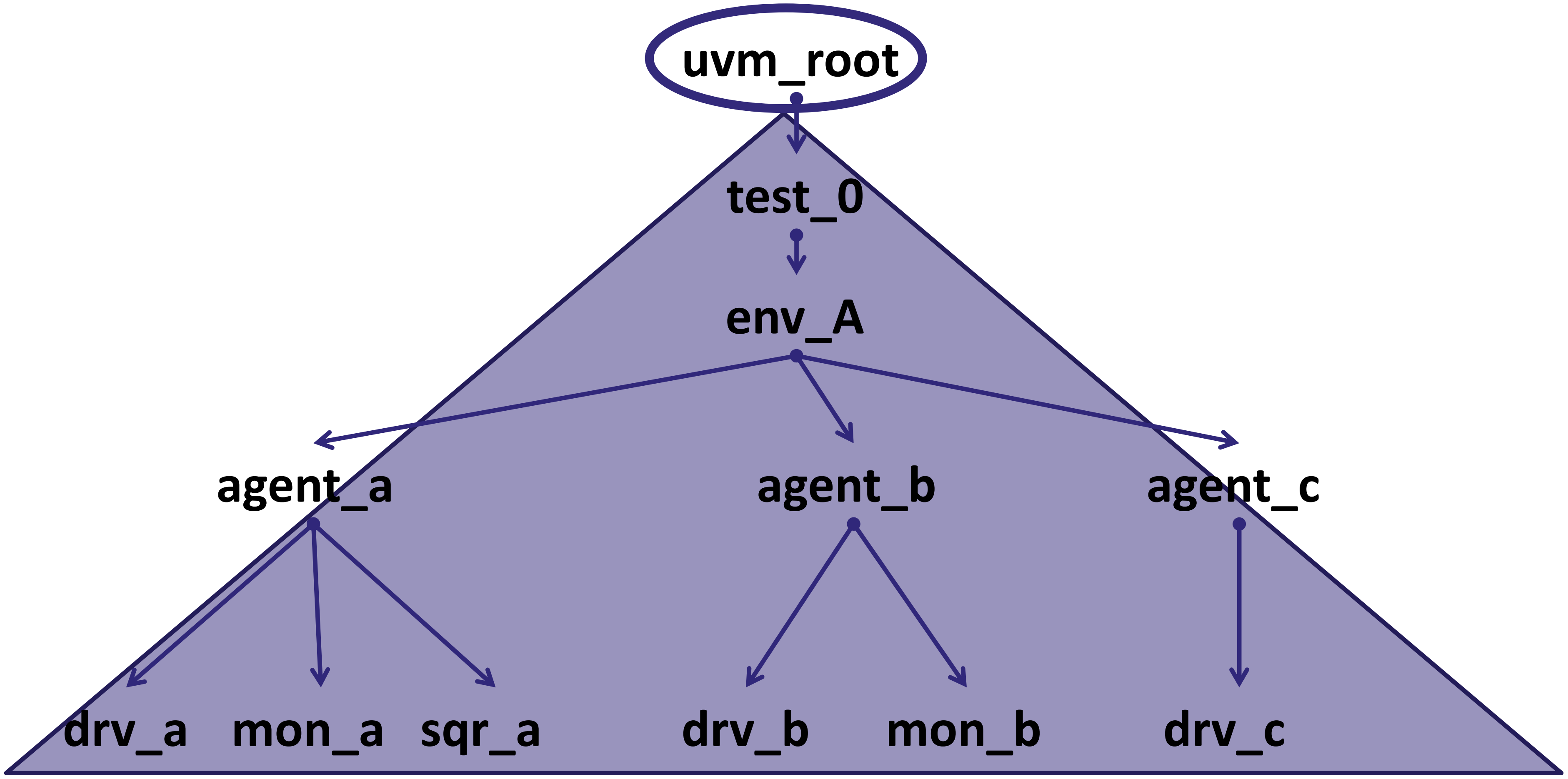
- Master component implements main phase
  - Wait for all objections dropped up to uvm_root

Monitoring Objections



Test bench Component Instance Hierarchy

Monitoring Objections

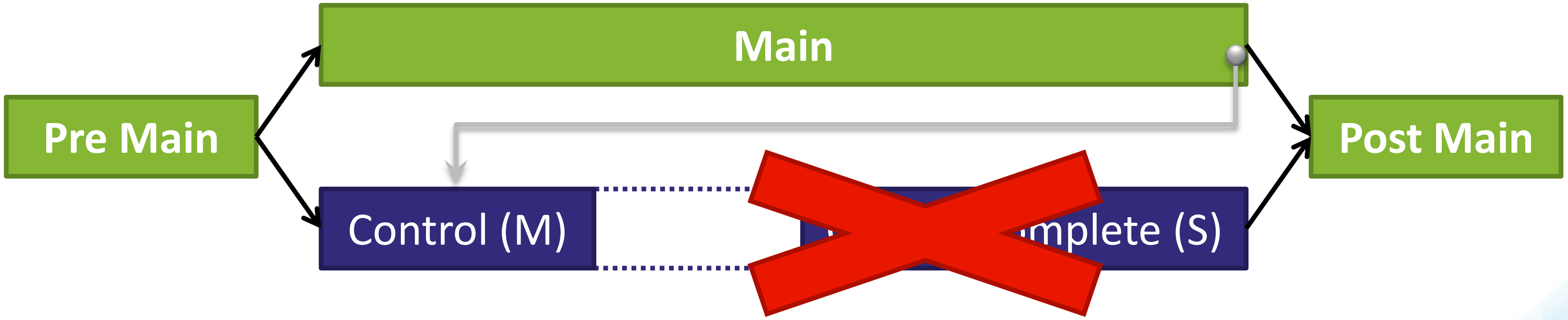


Test bench Component Instance Hierarchy

Solution to issue #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



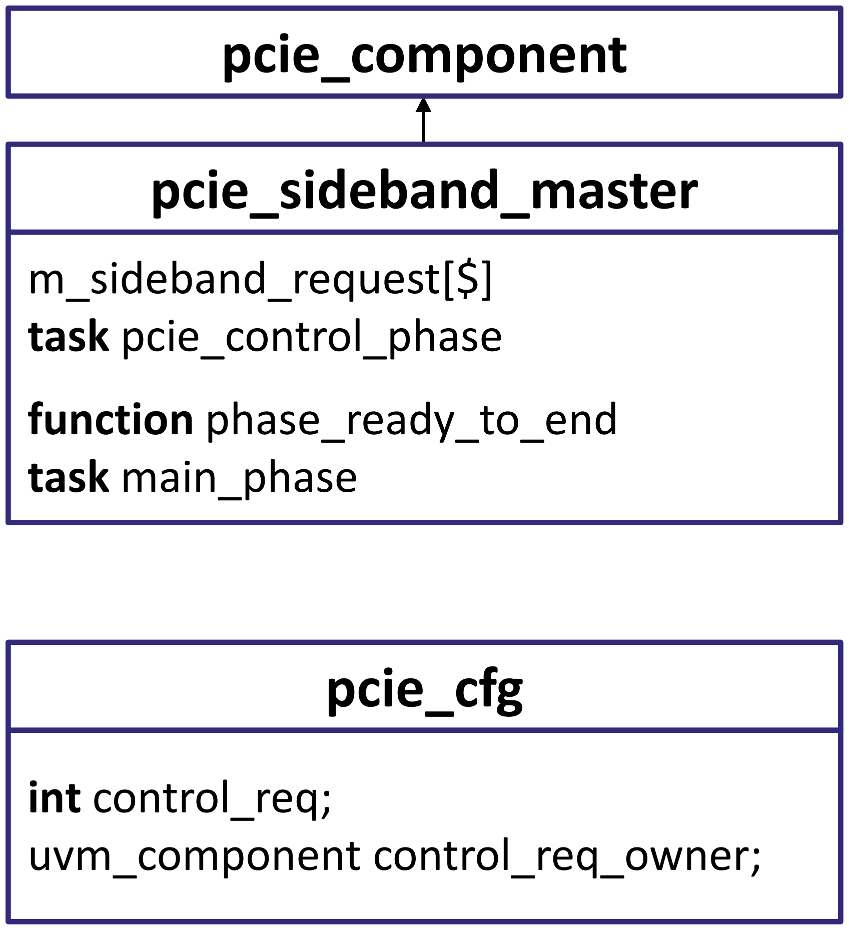
Solution to issue #2

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

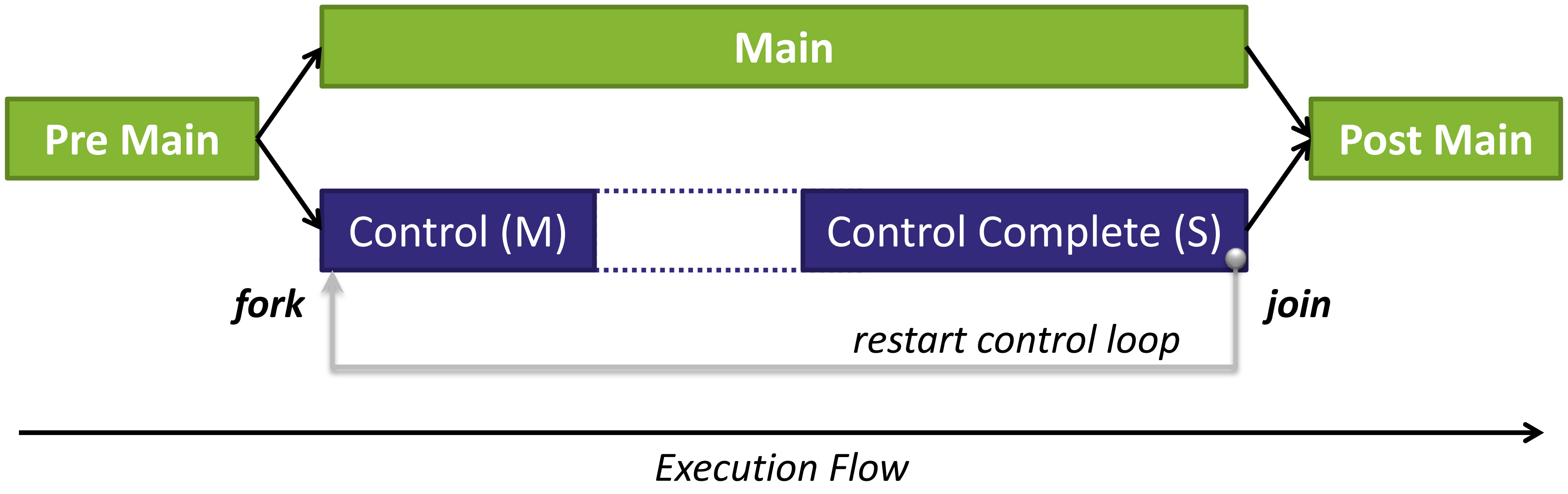
Master Component



- Is a pcie_component
- Processes requests
  - pcie_control_phase
- Posts to bulletin board
- Processes phase jumps
  - phase_ready_to_end
- Monitor main phase for phase exit

Custom Phasing Issue #3

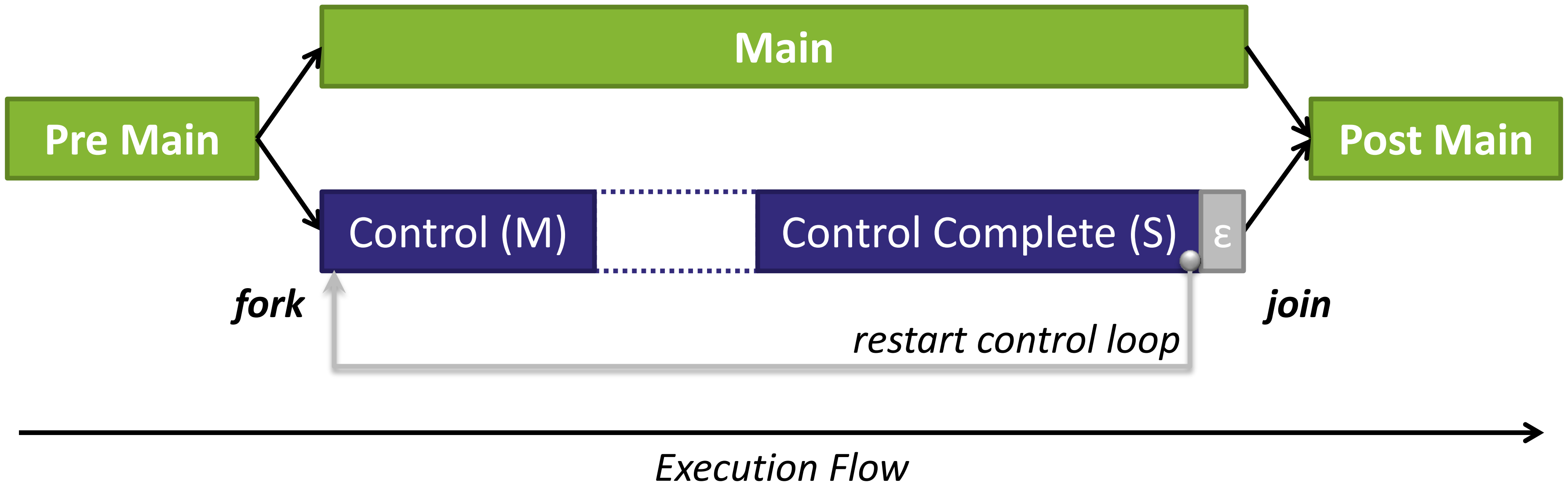
Control Complete Exit



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

Solution to issue #3

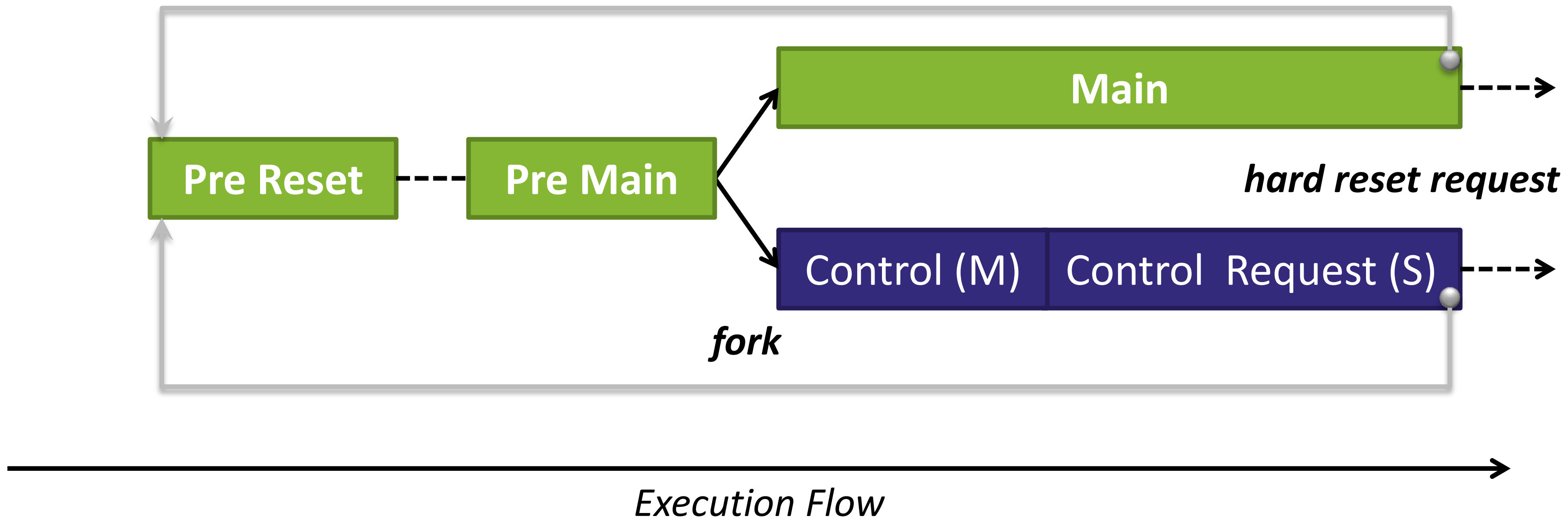
Terminal Phase



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

Custom Phasing Issue #4

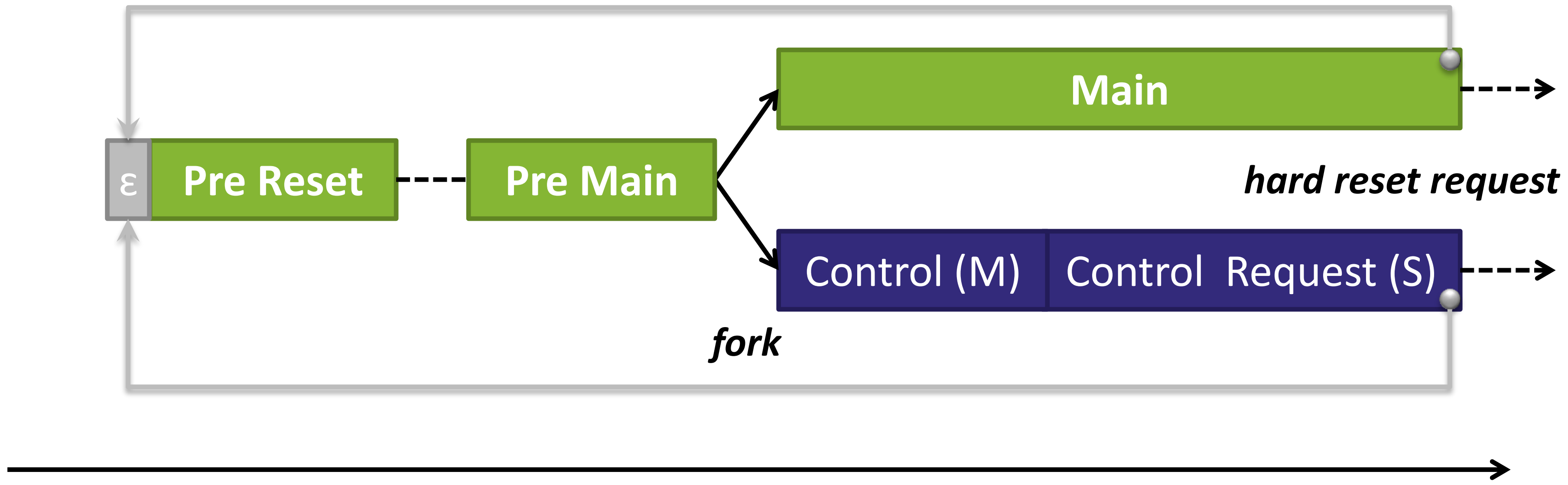
Multiple Phase Executions



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

Solution to Issue #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
```

Master Phase Jumping

```
class pcie_sideband_master extends pcie_component;
  function phase_ready_to_end(uvm_phase ph);
    uvm_phase imp = ph.get_imp();









  endfunction
endclass
```

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());




  endfunction
endclass
```

- Local jump with phase reference

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

Agenda

Results

- Gracefully handle two kinds of global events
  - Catastrophic -- hard reset, re-configuration
  - Non-catastrophic -- quiescence

- Used for ~1.5 years

- Have taped-out 2 customer programs
  - Ongoing project

Conclusions

- Custom phasing is viable for global event handling
  - passive, push-style notification
  - synchronization with handshake

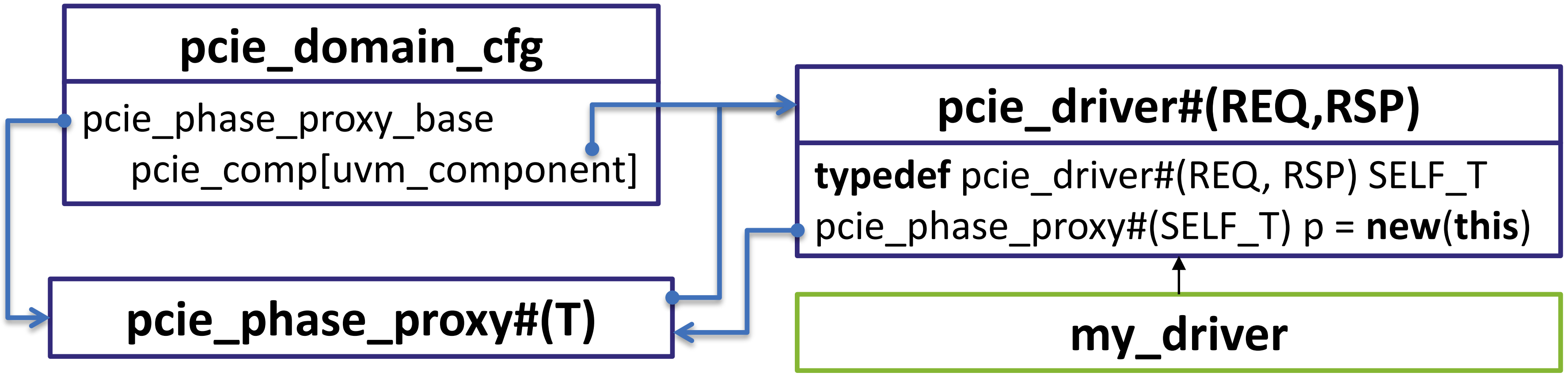
- Setup of custom phasing is fairly straightforward
  - Phase proxies to handle type-parameterized components

- Phasing hijinks solved with dummy phases

- Details in the paper!

Thank You

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