



Wait a sec, I'm not done! Delay `$finish` from every ``uvm_fatal` report message

Jeremy Ridgeway

June 30, 2020



Agenda

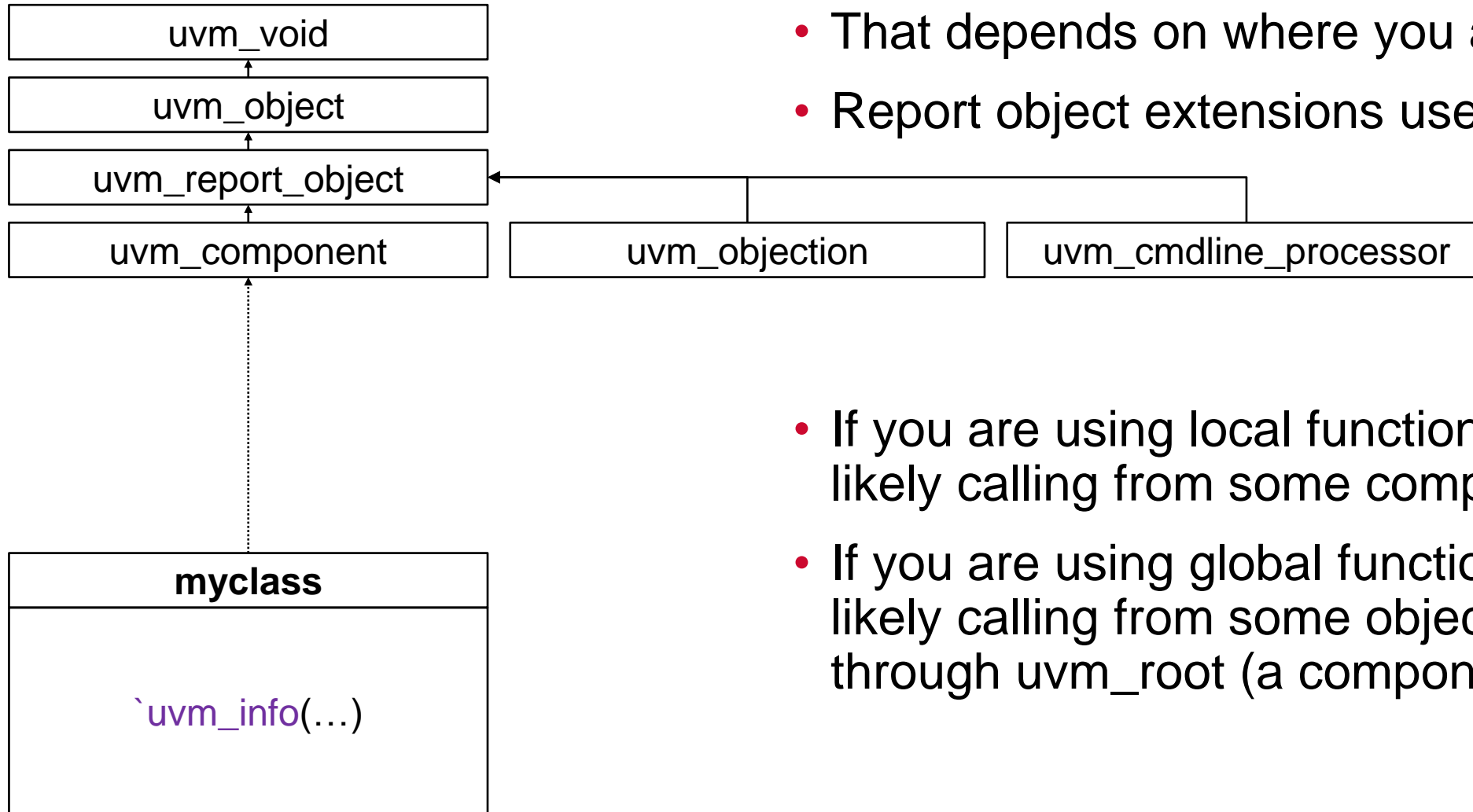
- Assumptions
- Catching all `uvm_fatal in UVM-1.1d
- Changes for UVM-1.2 / UVM-IEEE

cvm = custom verification methodology

Assumptions

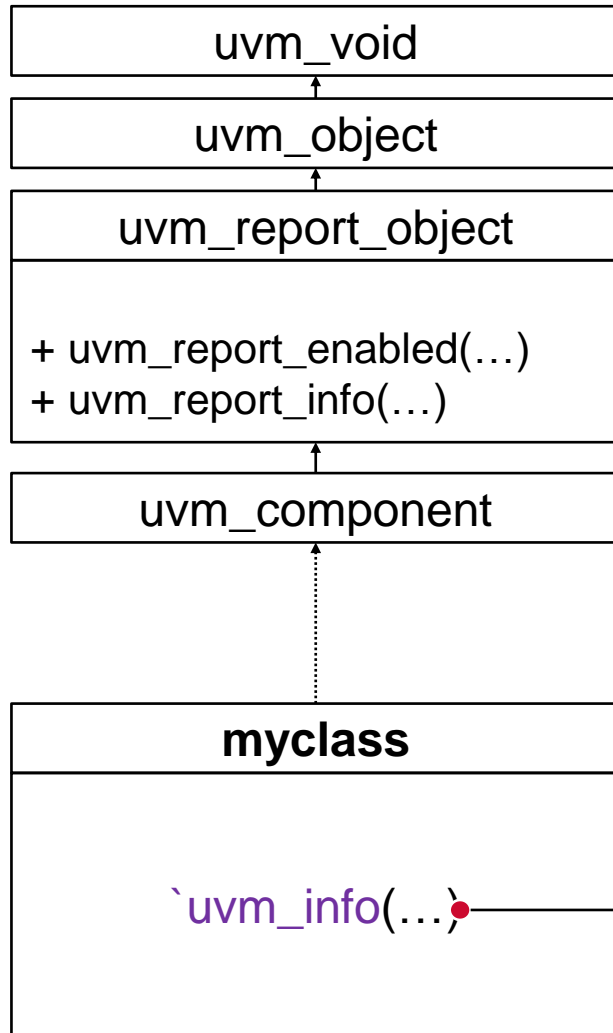
- You can override the report server
- There exists a single report server in the simulation

`uvm_info("ID", "message", UVM_LOW) – what happens?



- That depends on where you are
- Report object extensions use local functions
- If you are using local functions then you are likely calling from some component
- If you are using global functions then you are likely calling from some object and going through `uvm_root` (a component)

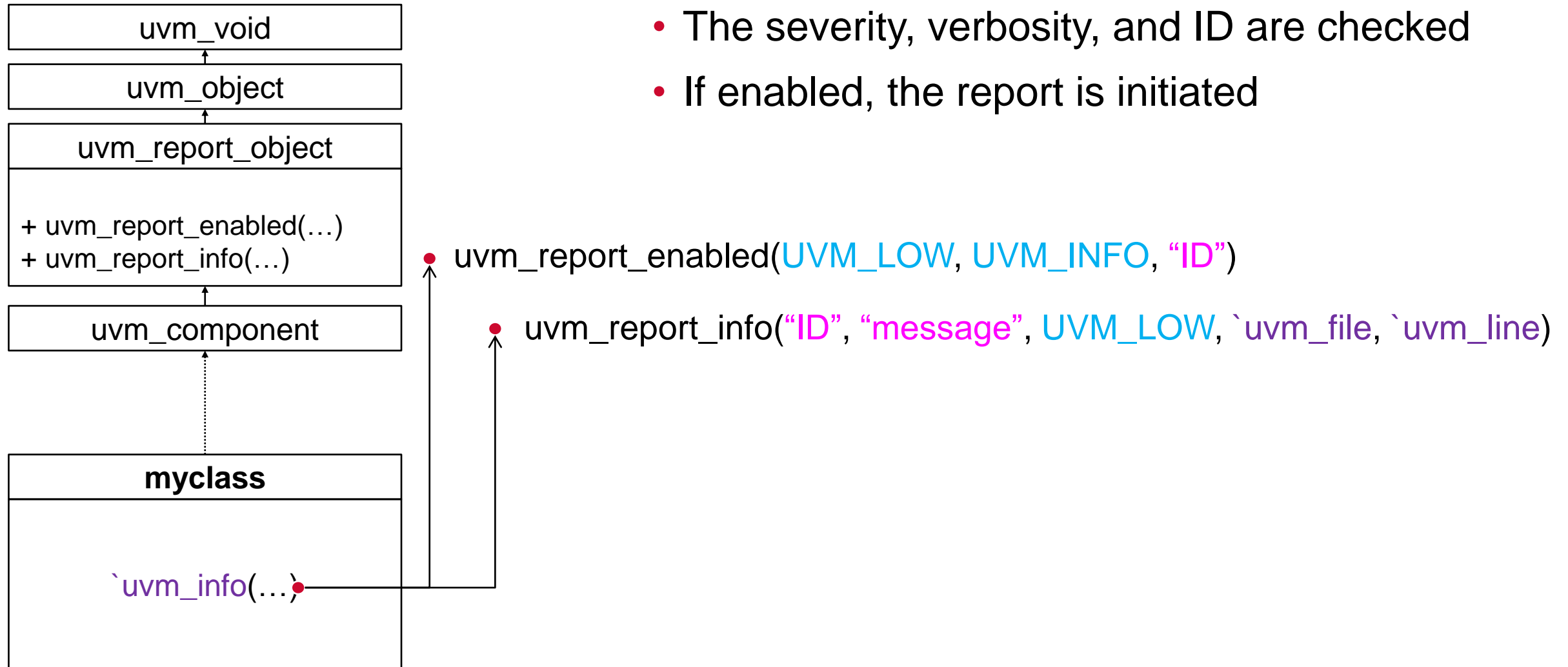
`uvm_info("ID", "message", UVM_LOW) – what happens?



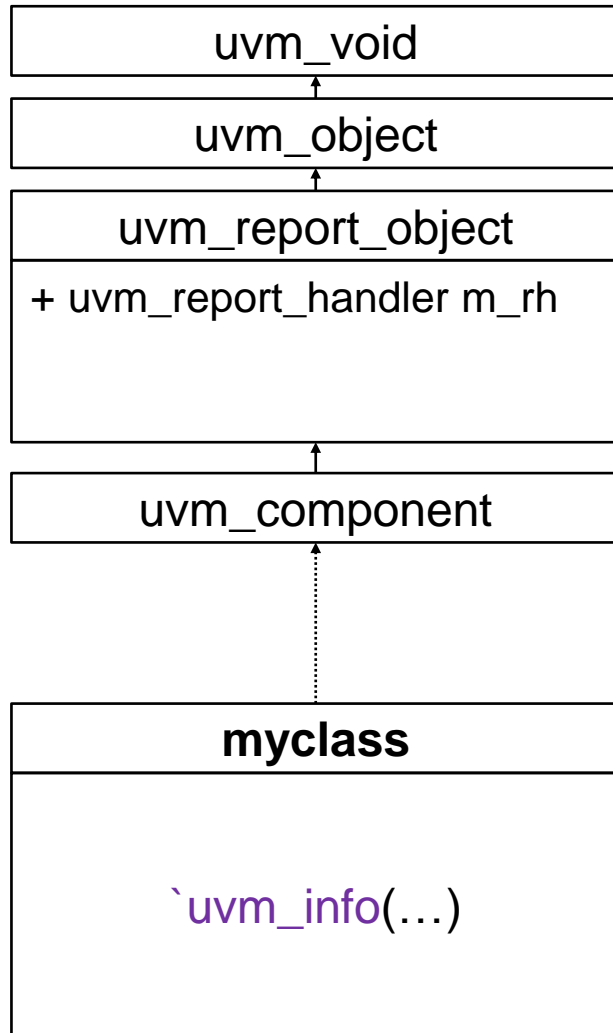
- The severity, verbosity, and ID are checked

• `uvm_report_enabled(UVM_LOW, UVM_INFO, "ID")`

`uvm_info("ID", "message", UVM_LOW) – what happens?

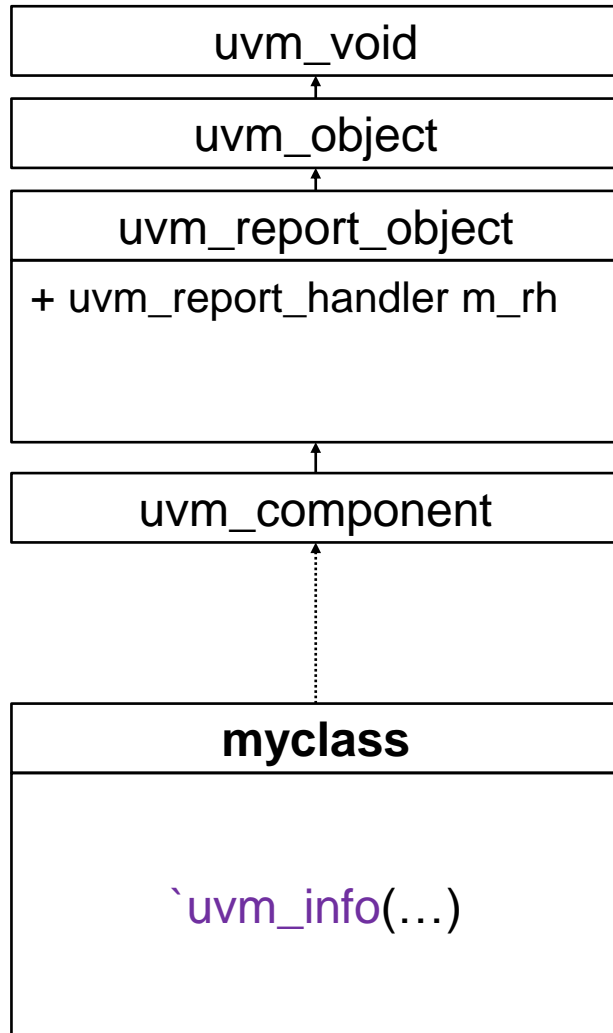


`uvm_info("ID", "message", UVM_LOW) – what happens?



- The report handler has no parent class
- `m_rh.report(.severity(UVM_INFO), .name(get_full_name()), .id("ID"), .message("message"), .verbosity_level(UVM_LOW), .filename(`uvm_file), .line(`uvm_line), client(this))`

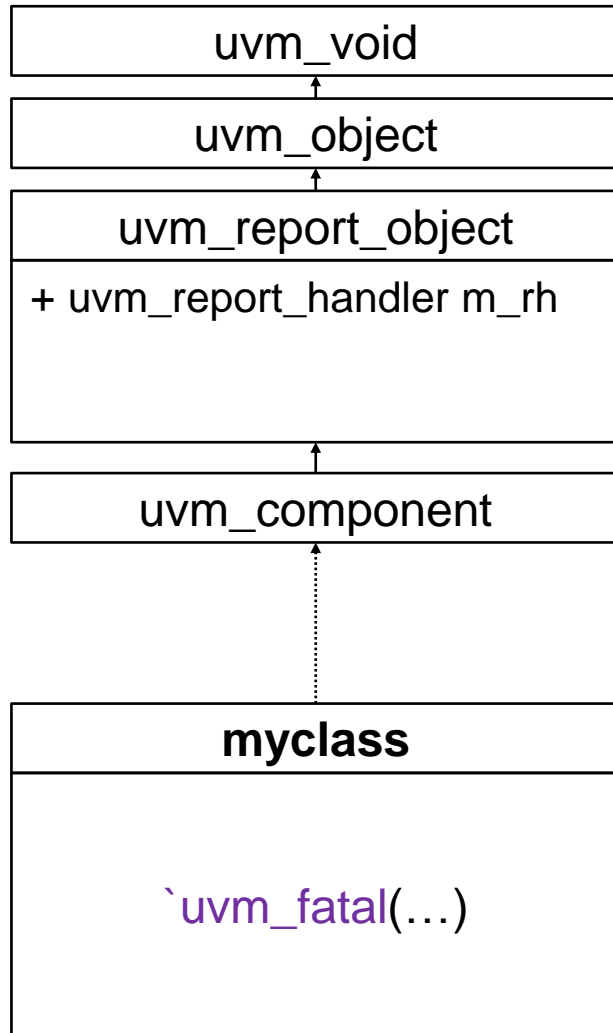
`uvm_info("ID", "message", UVM_LOW) – what happens?



- The report handler has no parent class
- The handler employs the report server
- `m_rh.report(.severity(UVM_INFO), .name(get_full_name()), .id("ID"), .message("message"), .verbosity_level(UVM_LOW), .filename(`uvm_file), .line(`uvm_line), client(this))`

```
uvm_report_server svr = uvm_report_server::get_server();  
  
svr.report(severity, name, id, message, verbosity_level,  
          filename, line, client);
```

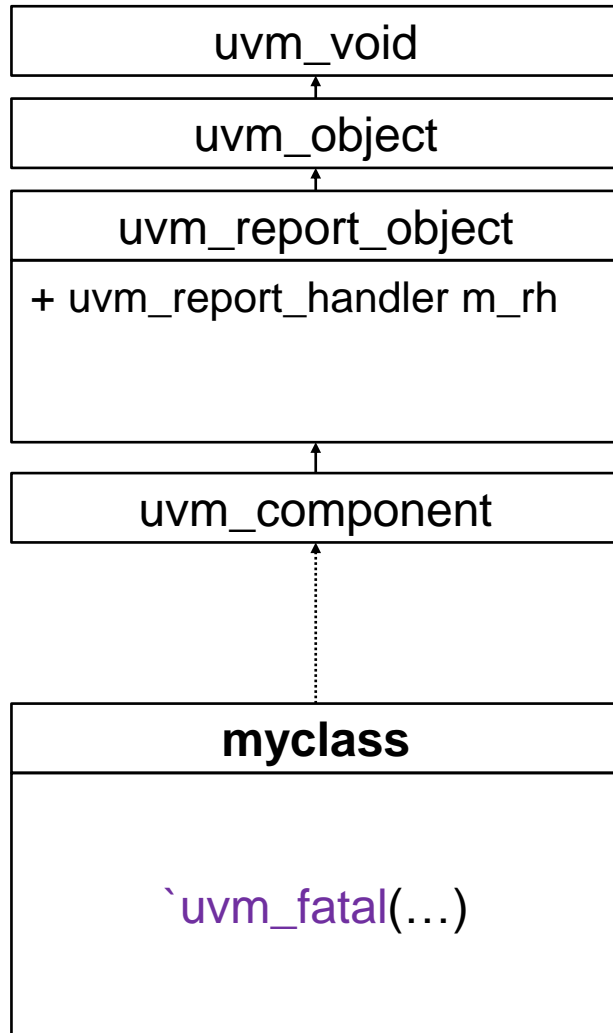

`uvm_fatal("ID", "message") – what happens?



- The report handler has no parent class
- The handler employs the report server
- `m_rh.report(.severity(UVM_FATAL), .name(get_full_name()), .id("ID"), .message("message"), .verbosity_level(UVM_NONE), .filename(`uvm_file), .line(`uvm_line), client(this))`

```
uvm_report_server svr = uvm_report_server::get_server();  
  
svr.report(severity, name, id, message, verbosity_level,  
          filename, line, client);
```

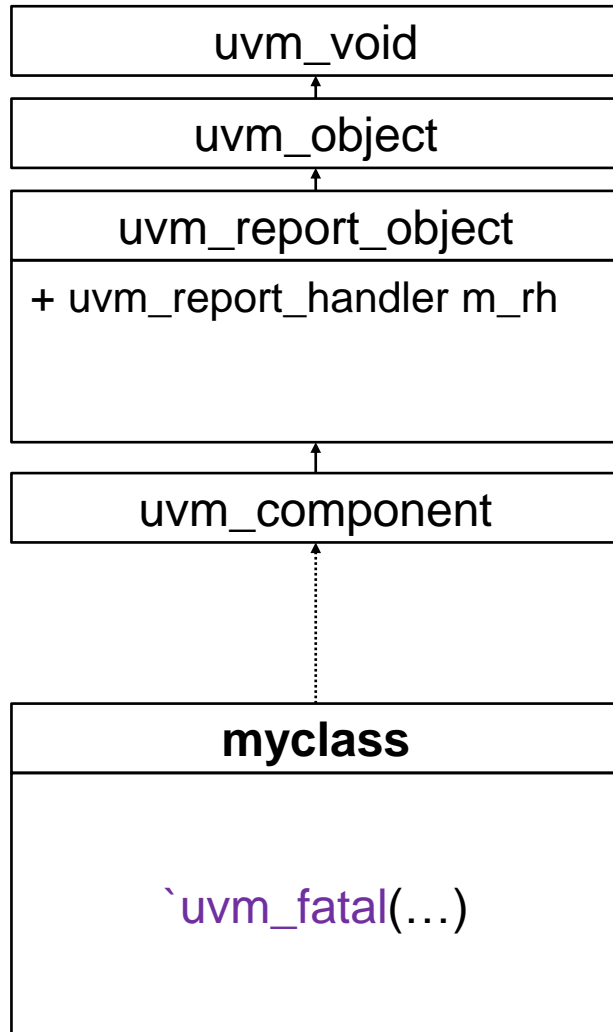
`uvm_fatal("ID", "message") – what happens?



- The report handler has no parent class
- The handler employs the report server
- We must override the client to intercept a fatal
- `m_rh.report(.severity(UVM_FATAL), .name(get_full_name()), .id("ID"), .message("message"), .verbosity_level(UVM_NONE), .filename(`uvm_file), .line(`uvm_line), client(this))`

```
uvm_report_server svr = uvm_report_server::get_server();  
  
svr.report(severity, name, id, message, verbosity_level,  
          filename, line, client)
```

`uvm_fatal("ID", "message") – what happens?



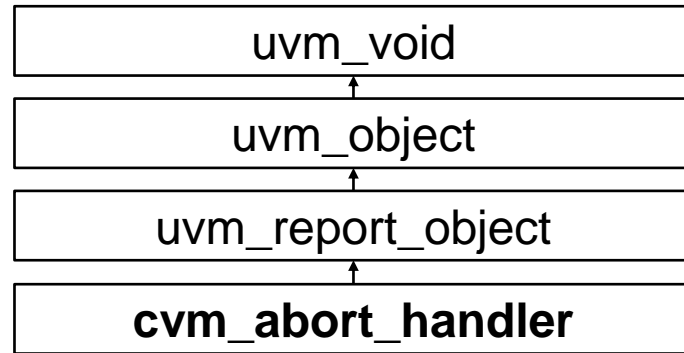
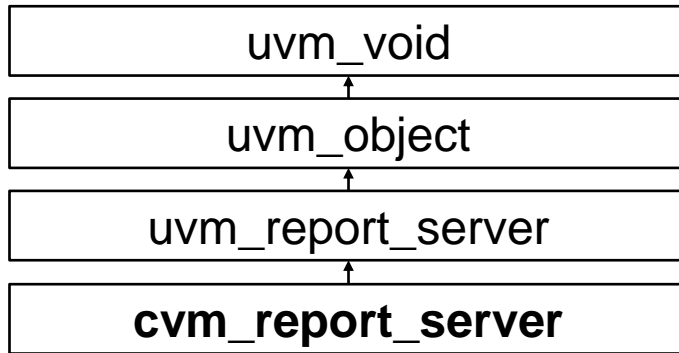
- The report handler has no parent class
 - The handler employs the report server
 - We must override the client to intercept a fatal
- `m_rh.report(.severity(UVM_FATAL), .name(get_full_name()), .id("ID"), .message("message"), .verbosity_level(UVM_NONE), .filename(`uvm_file), .line(`uvm_line), client(this))`

```
uvm_report_server svr = uvm_report_server::get_server();
```

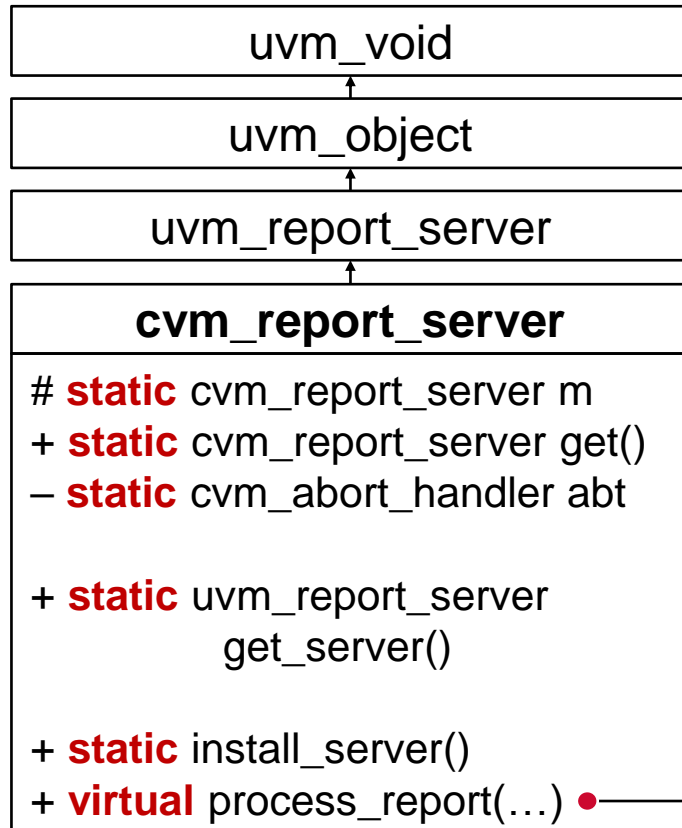
```
svr.report(severity, name, id, message, verbosity_level,  
          filename, line, client)
```

```
if(action & UVM_EXIT) client.die()
```

`uvm_fatal override architecture



Custom Report Server



- Any report with action of **UVM_EXIT**, override with the report handler
- Abort handler is a singleton

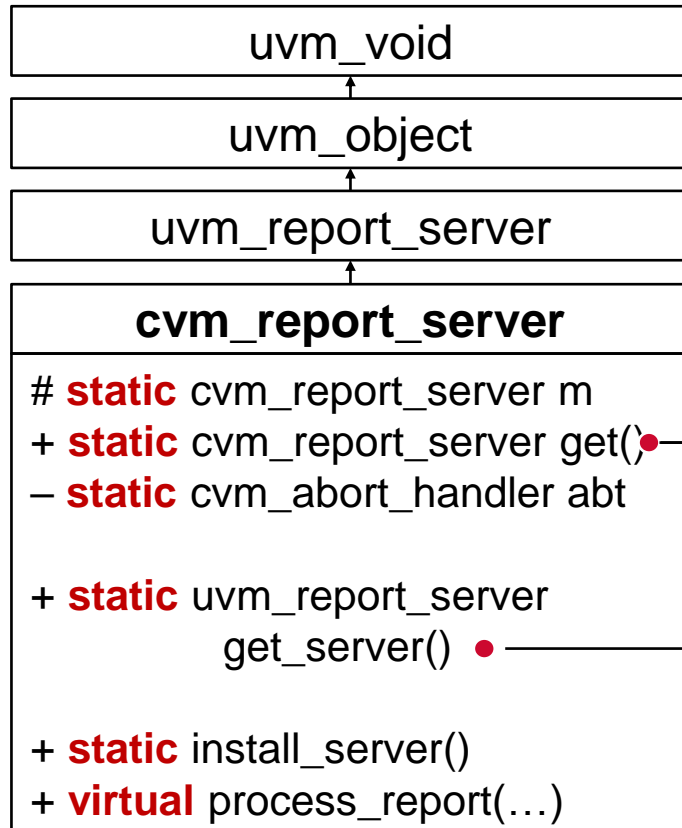
```
function void process_report(uvm_severity severity, string name,
  string id, string message, uvm_action action,
  UVM_FILE file, string filename, int line,
  string composed_message, int verbosity_level,
  uvm_report_object client); // override the client

if((action & UVM_EXIT) && (abt != null))
  client = abt;

super.process_report(severity, name, id, message, action, file,
  filename, line, composed_message,
  verbosity_level, client);

endfunction
```

Custom Report Server



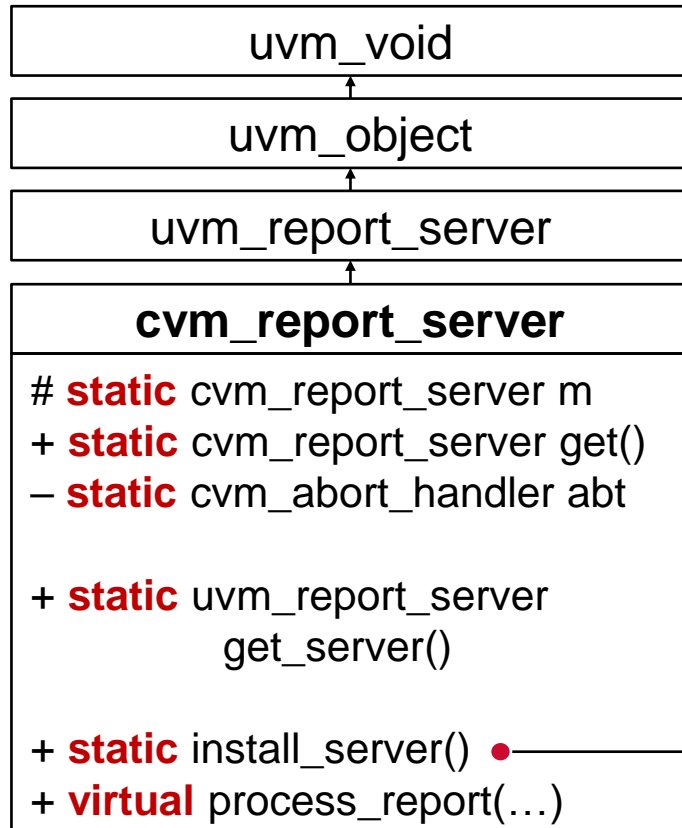
- Assume server is a singleton

```
function new(); // outside of factory
    super.new();
    abt = my_abort_handler::get();
endfunction

function cvm_report_server get();
    if(m == null) m = new();
    return m;
endfunction

function uvm_report_server get_server();
    return get(); // required to tie into UVM
endfunction
```

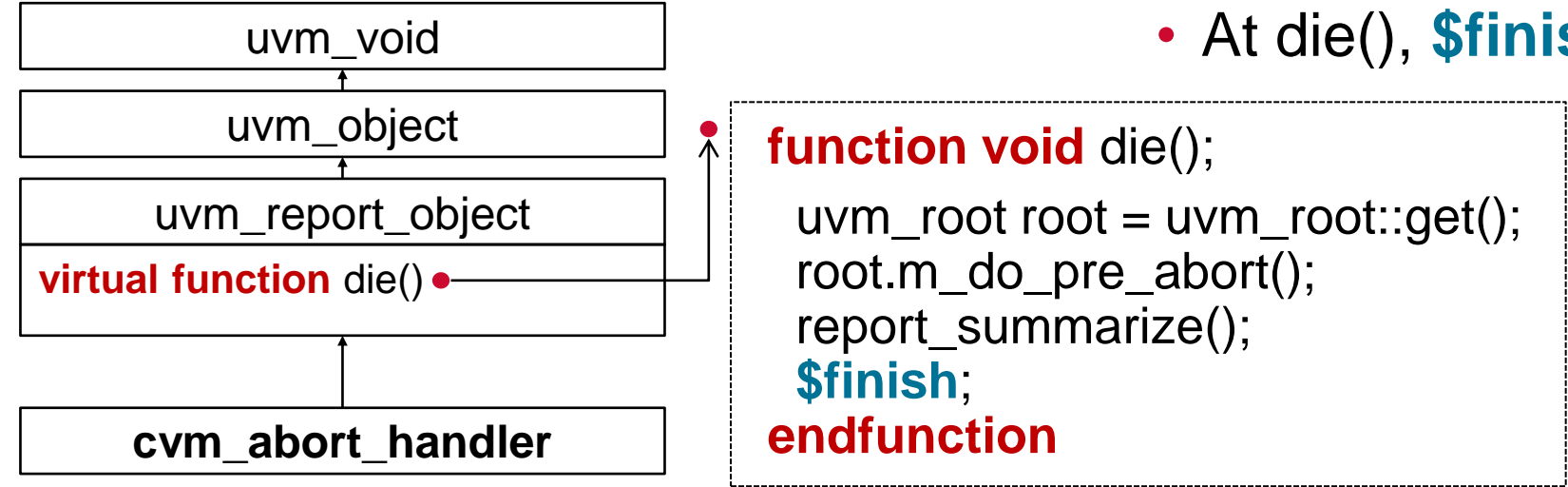
Custom Report Server



- Installation of custom server is manual
 - We do this in base test build_phase

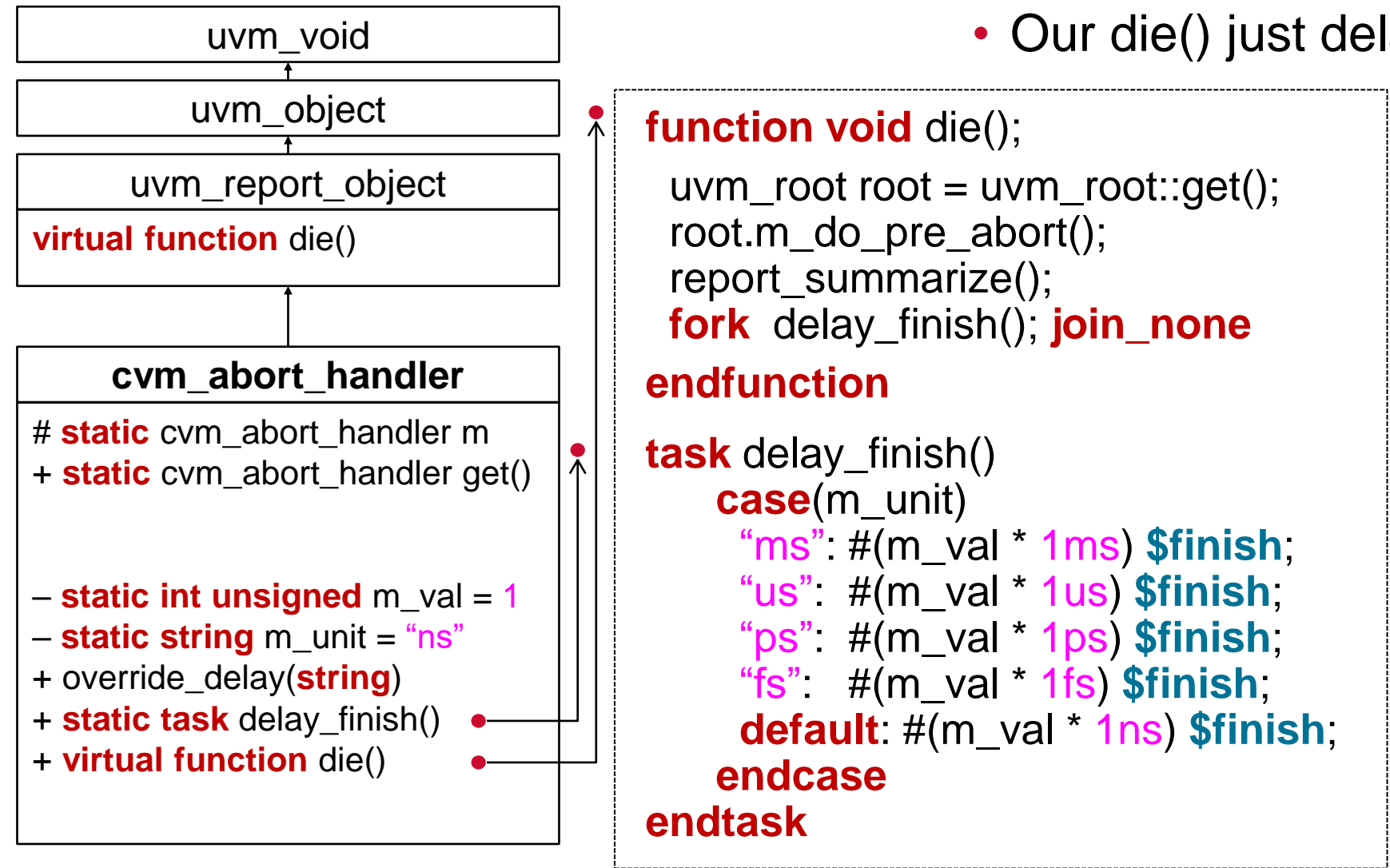
```
function void install_server();  
    uvm_report_server cvm_srvr = cvm_report_server::get_server();  
    uvm_report_server uvm_srvr = uvm_report_server::get_server();  
    if(cvm_srvr != uvm_srvr) begin  
        uvm_report_server::set_server(cvm_srvr);  
        `uvm_info("CVM_RPT_SRVR",  
                "Custom report server installed", UVM_NONE)  
    end  
endfunction
```

UVM Abort Handler

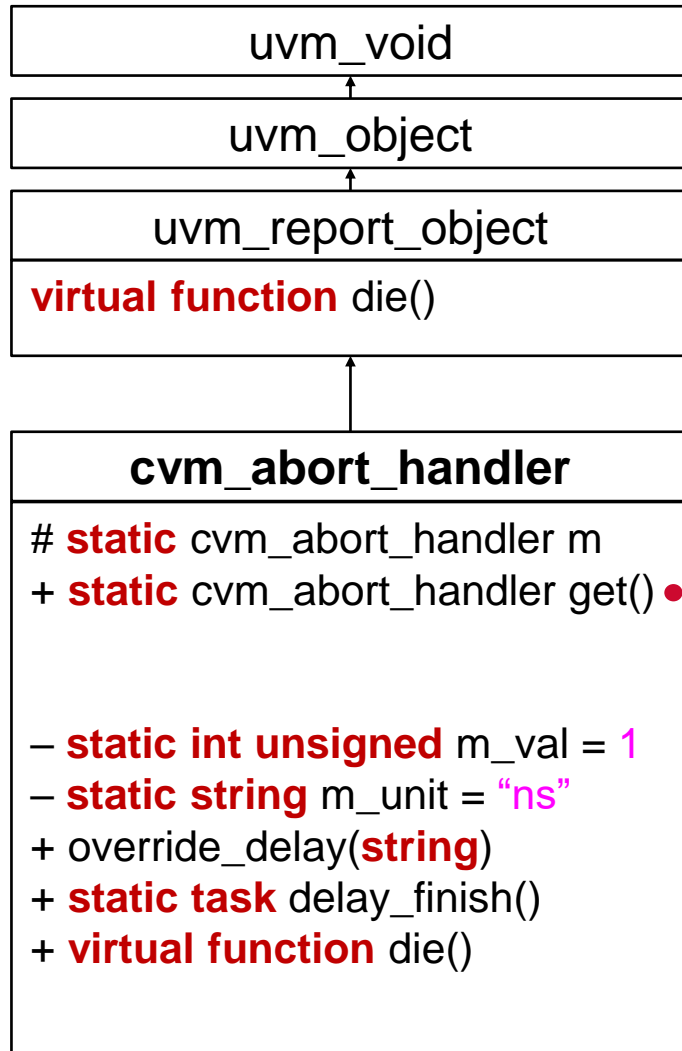


- At die(), **\$finish** is called immediately

Custom Abort Handler



Custom Abort Handler



- Assume singleton object instance

```
`uvm_object_utils(cvm_abort_handler)
function new(string name = "cvm_abort_handler");
    super.new(name); // but, don't use this new – look later
endfunction

function void cvm_abort_handler get();
    if(m == null)
        m = cvm_abort_handler::type_id::create("abt", null);
    return m;
endfunction
```

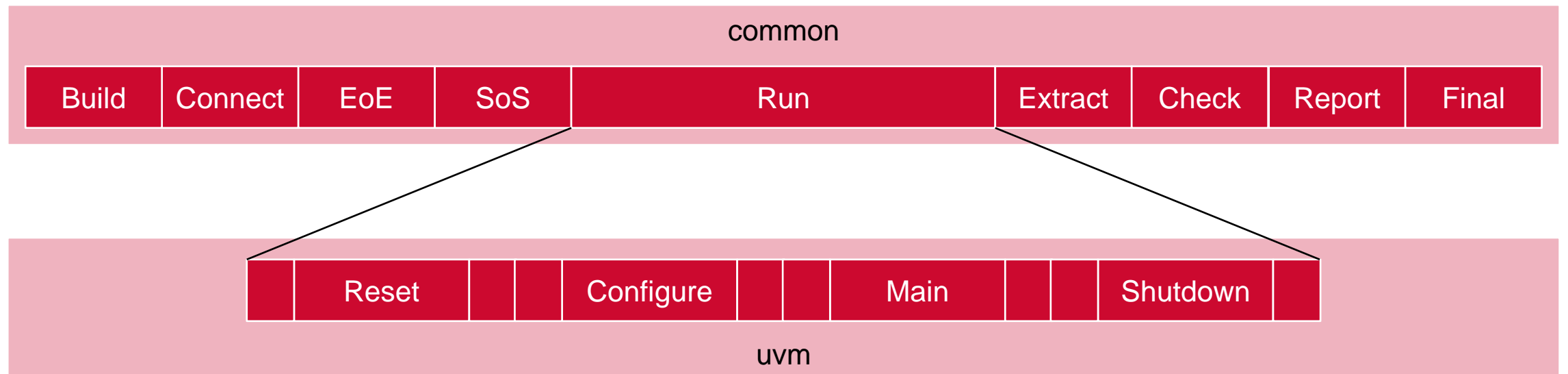
But, There's a Problem!

- This may or may not delay the simulation exit
- Phasing is the problem

Wait a sec, I'm not done!
Delay \$finish from **EVERY**
``uvm_fatal`` report message

Phasing

- Phases execute within domains (common, uvm)
- Domains may execute simultaneously



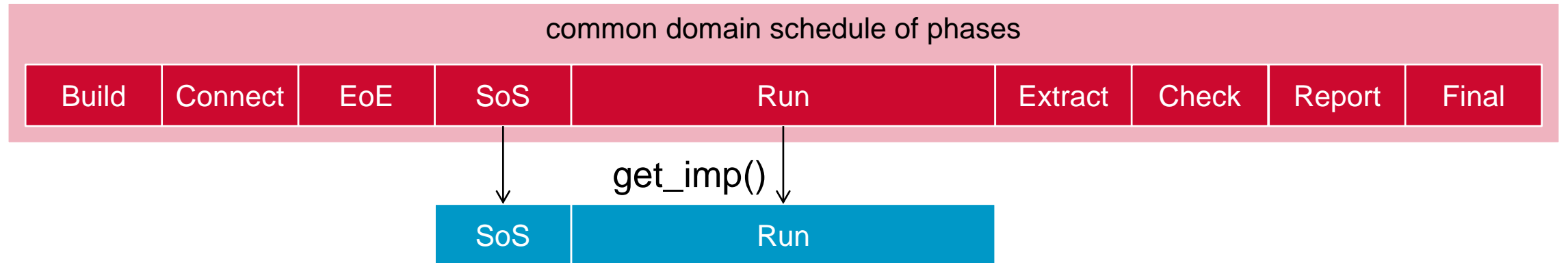
Phasing Schedule

- Phases are singletons, called an implementation phase



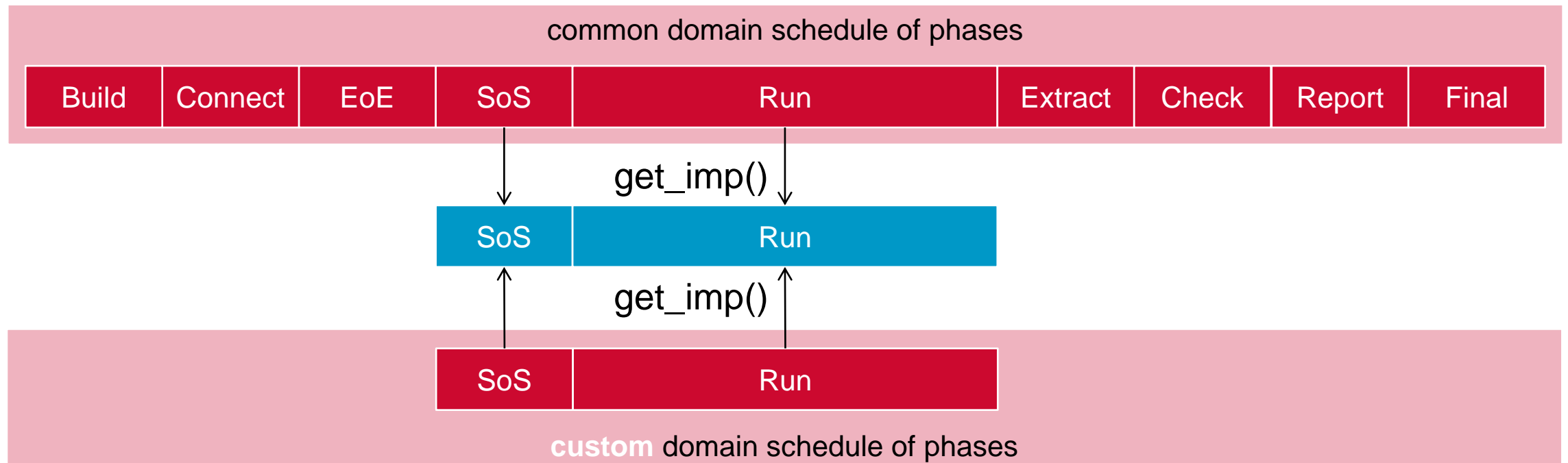
Phasing Schedule

- Phases are singletons, called an implementation phase
- Phase schedules contain phase representatives, called a node phase



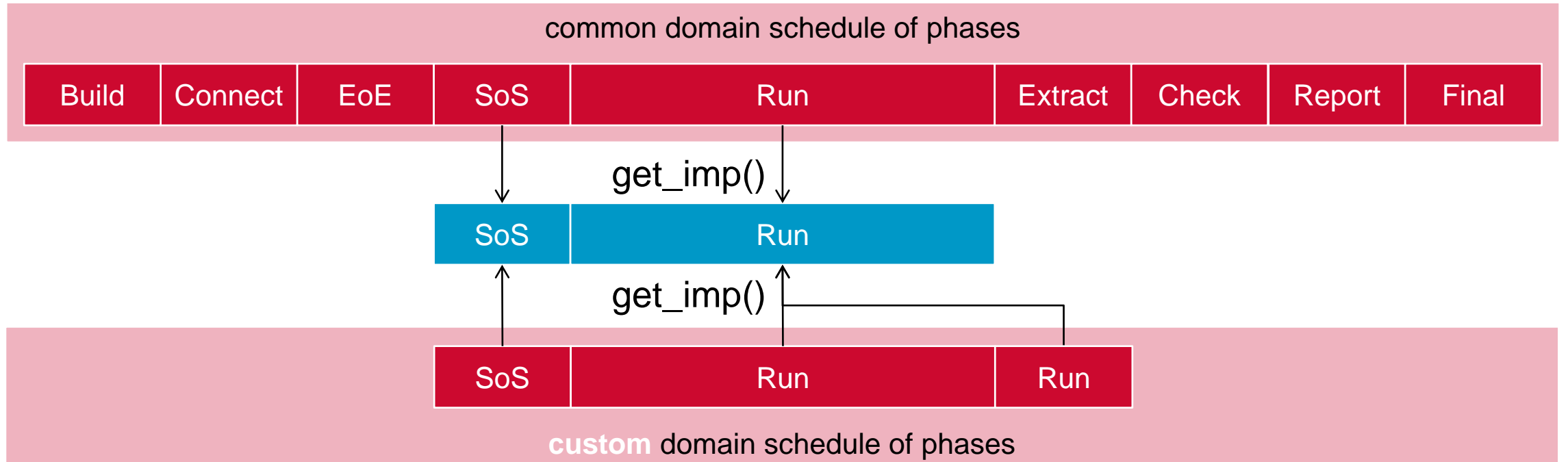
Phasing Schedule

- Phases are singletons, called an implementation phase
- Phase schedules contain phase representatives, called a node phase
 - A phase may occur in multiple schedules



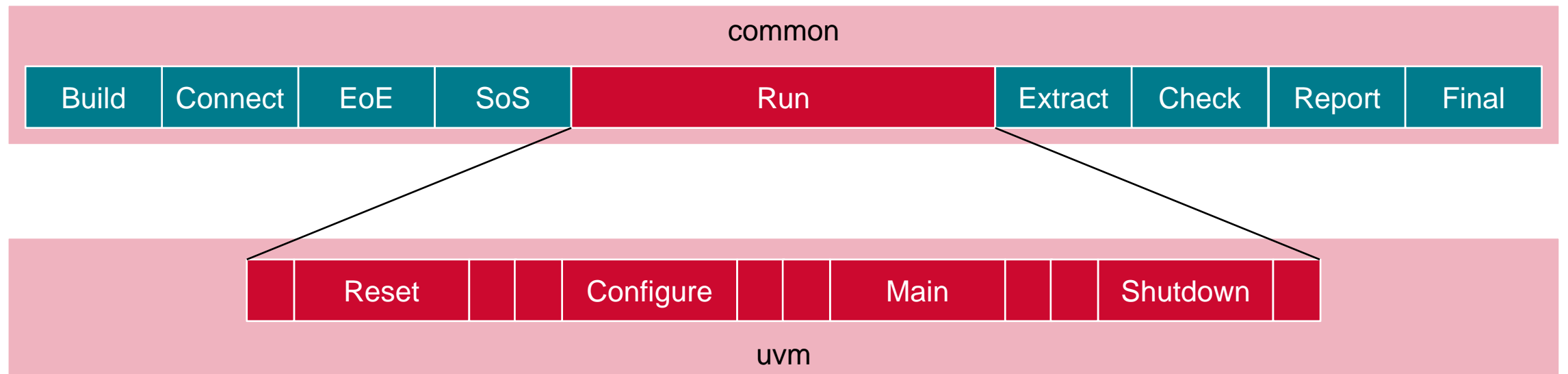
Phasing Schedule

- Phases are singletons, called an implementation phase
- Phase schedules contain phase representatives, called a node phase
 - A phase may occur in multiple schedules
 - A phase may occur multiple times in a single schedule



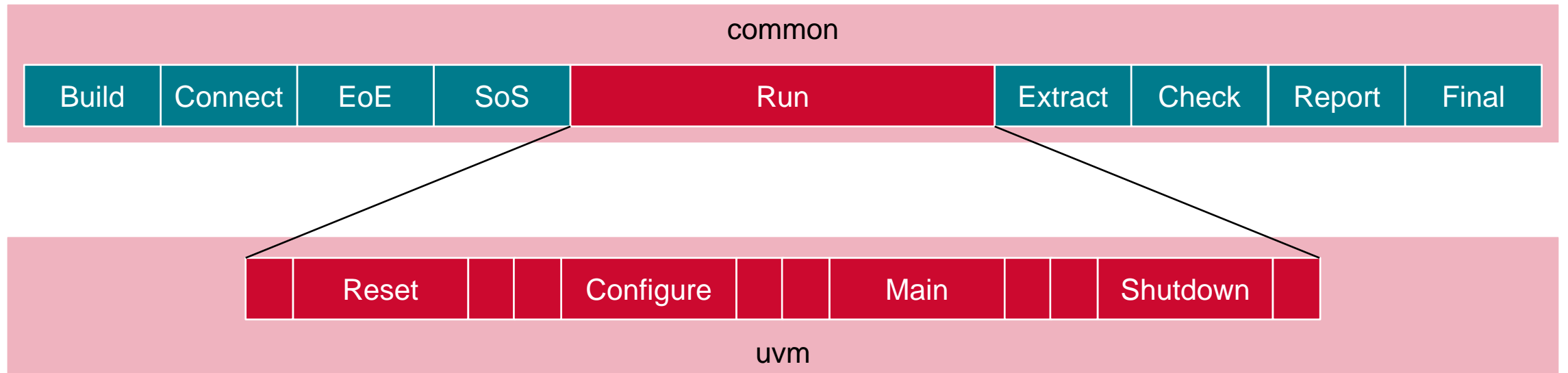
Phasing

- Some phases take no time (**functions**)
- Others may consume simulation time (**tasks**)



Phasing

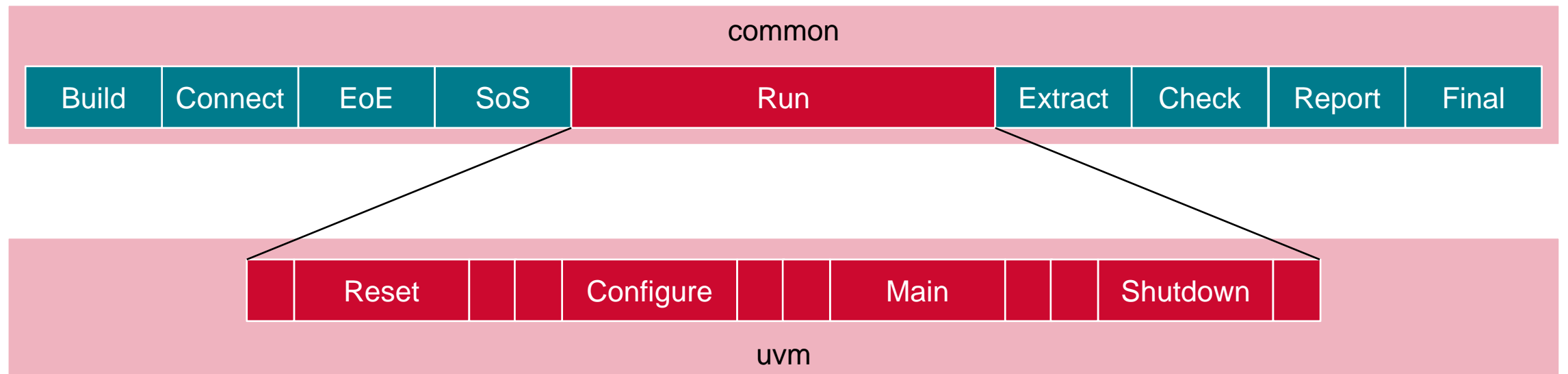
- Some phases take no time (**functions**)
- Others may consume simulation time (**tasks**)



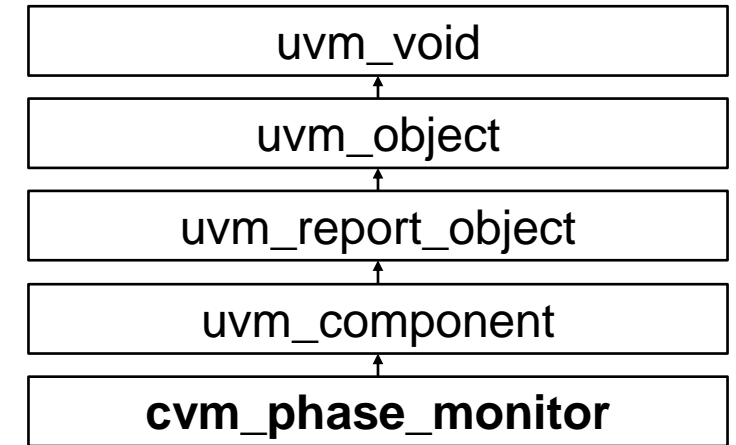
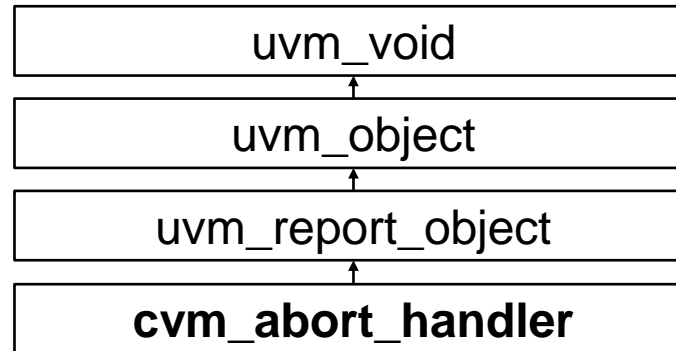
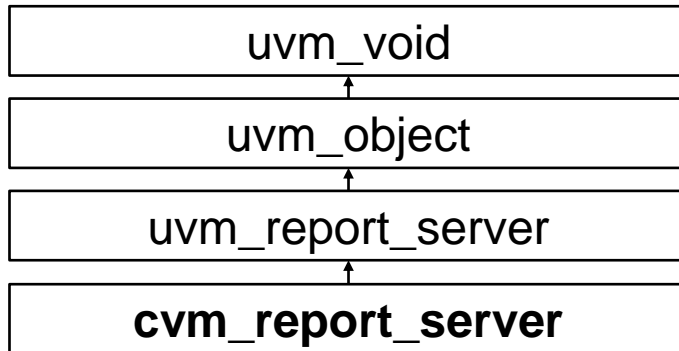
Cannot object in function phases

How to Know Which Phase is Active at `uvm_fatal`?

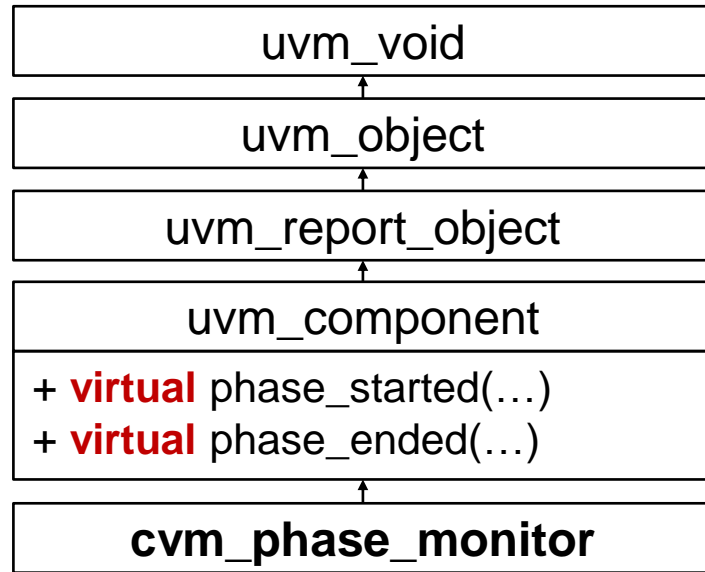
- No built-in function
- Must monitor phase changes via uvm_component



`uvm_fatal override architecture

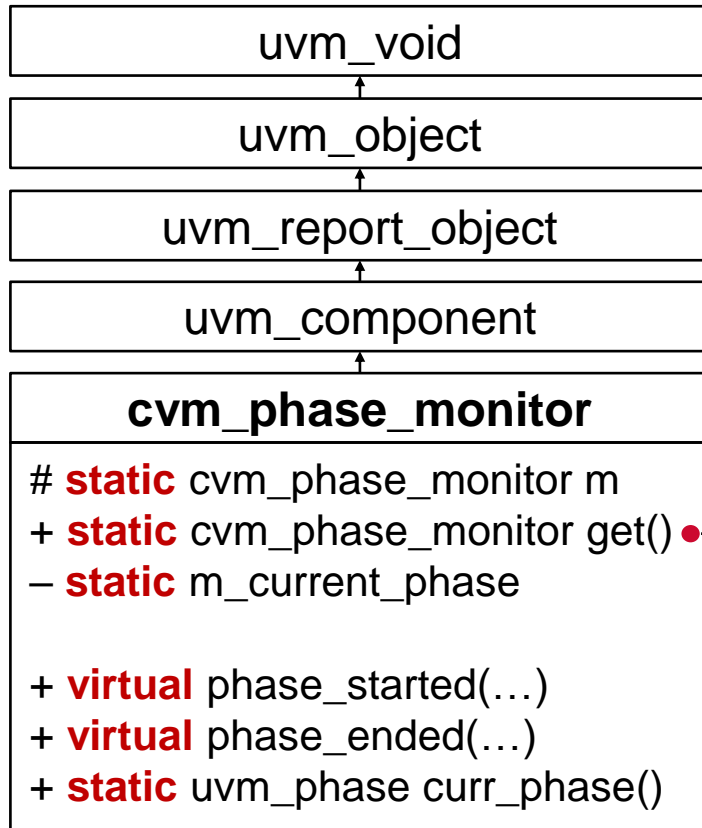


Phase Monitor



- Monitor phases via a uvm_component

Phase Monitor

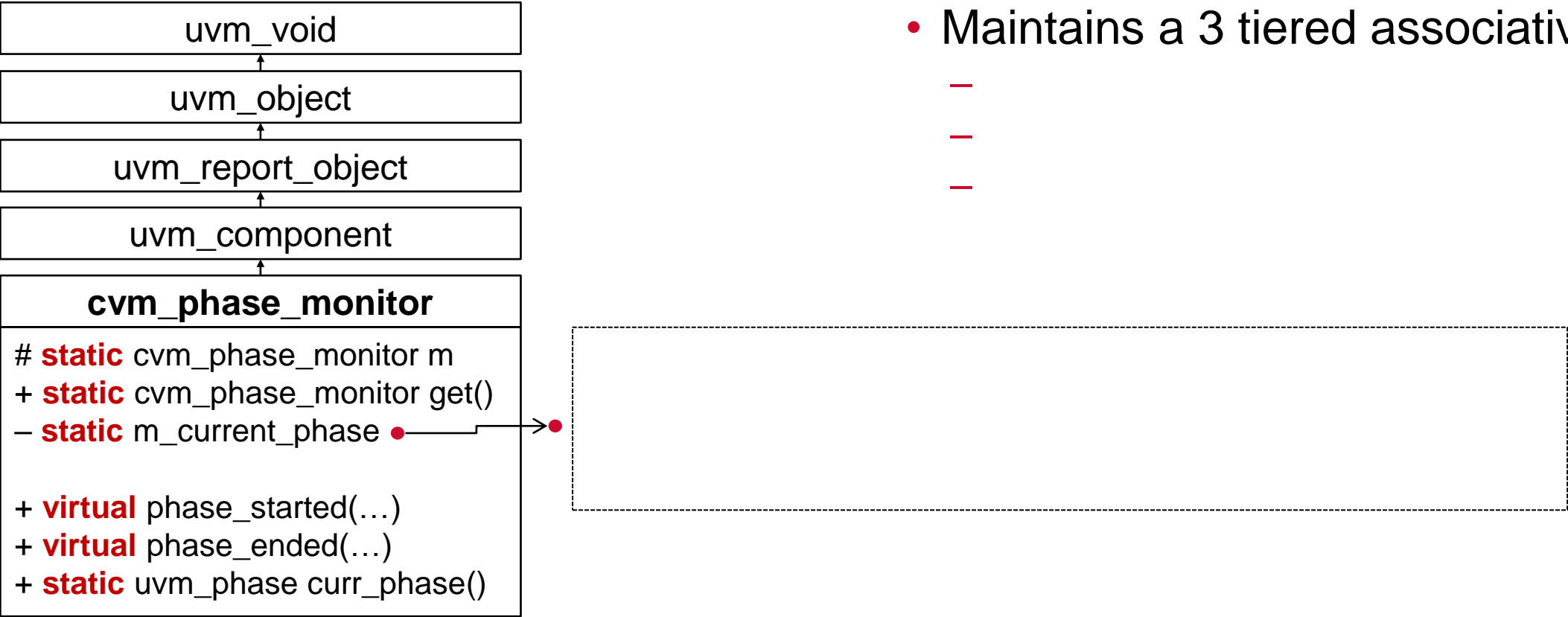


- Is a singleton instance

```
`uvm_component_utils(cvm_phase_monitor)
function new(string name = "cvm_phase_mon",
             uvm_component parent = null);
    super.new(name, parent);
endfunction

function cvm_phase_monitor get();
    if(m == null)
        m = cvm_phase_monitor::type_id::create("mon", uvm_top);
    return m;
endfunction
```

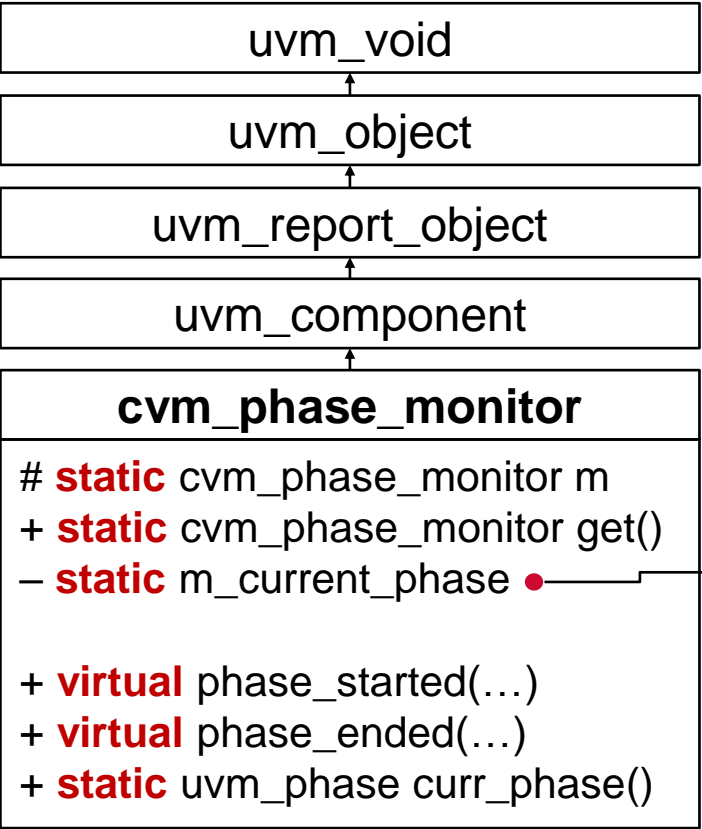
Phase Monitor



- Maintains a 3 tiered associative array

—
—
—

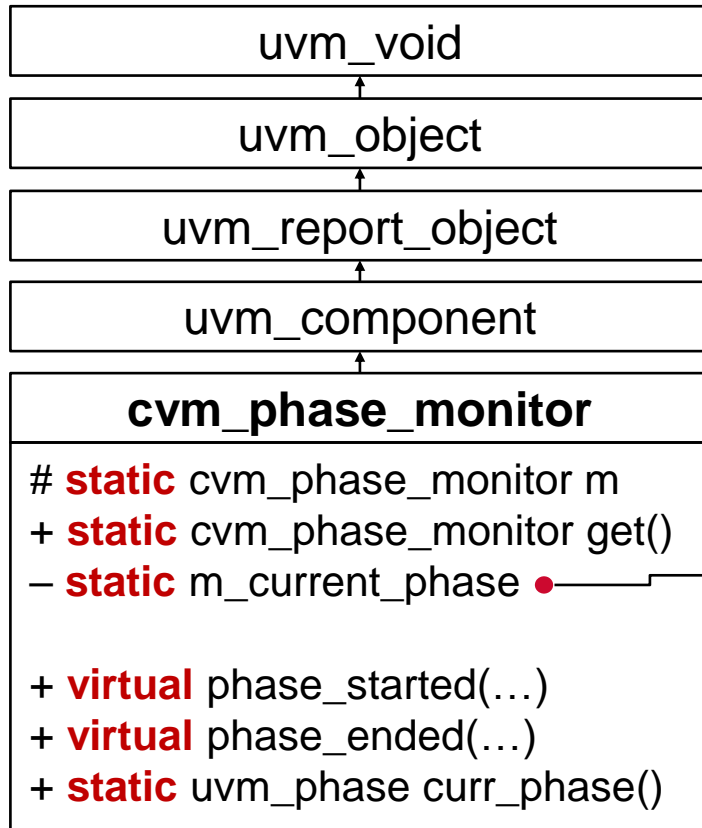
Phase Monitor



- Maintains a 3 tiered associative array
 -
 -
 - (3) maintain unique phases (imp |-> node)

typedef uvm_phase phase_map_t[uvm_phase] // (3)

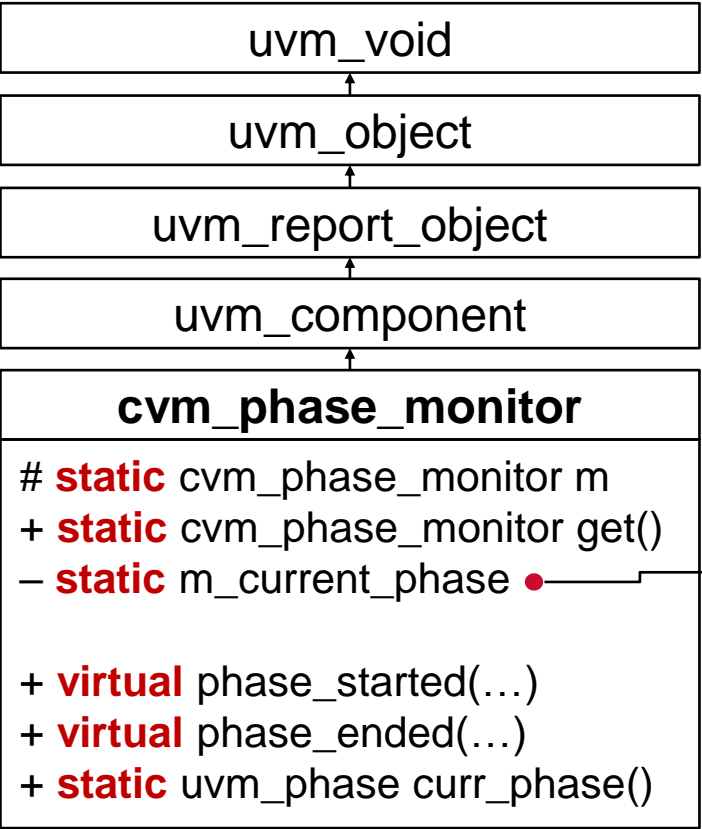
Phase Monitor



- Maintains a 3 tiered associative array
 -
 - (2) within unique uvm_domains
 - (3) maintain unique phases (imp |-> node)

```
typedef uvm_phase phase_map_t[uvm_phase] // (3)
typedef phase_map_t domain_map_t[uvm_domain] // (2)
```

Phase Monitor

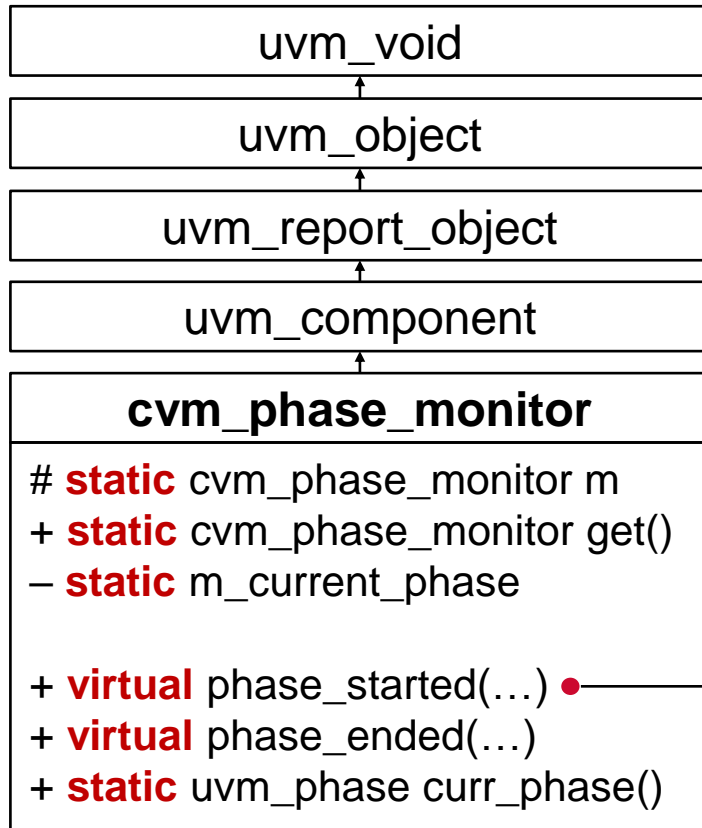


- Maintains a 3 tiered associative array
 - (1) ensure array per cvm_phase_monitor
 - (2) within unique uvm_domains
 - (3) maintain unique phases (imp |-> node)

```
typedef uvm_phase phase_map_t[uvm_phase] // (3)
typedef phase_map_t domain_map_t[uvm_domain] // (2)
domain_map_t m_current_phase[cvm_phase_monitor] // (1)
```

```
m_current_phase[this][domain][imp] = phase
```

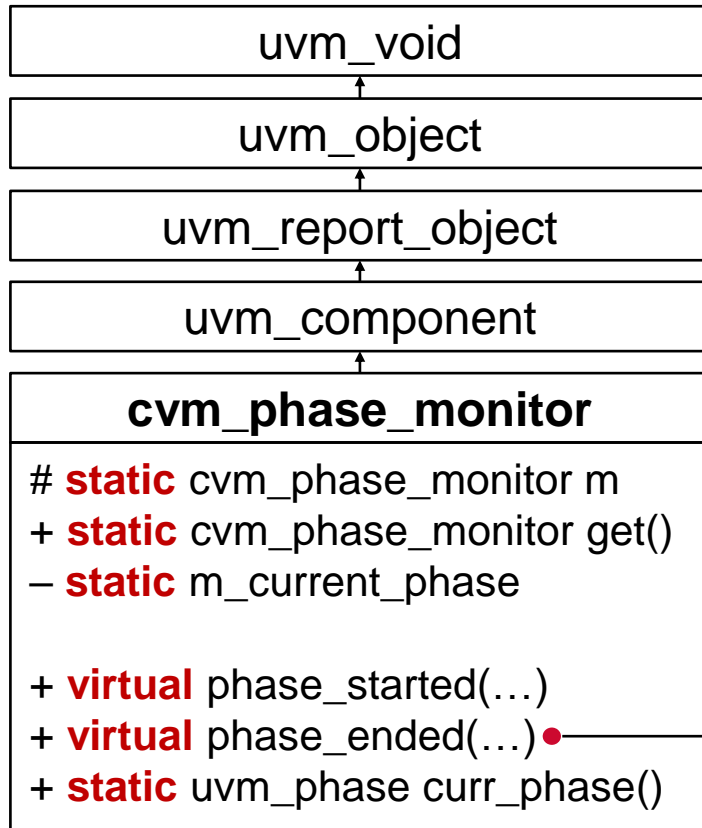
Phase Monitor



- **Add a phase at start**

```
function void phase_started(uvm_phase phase);
    uvm_domain dom = phase.get_domain();
    uvm_phase imp = phase.get_imp();
    if(!m_current_phase.exists(this)) begin
        domain_map_t mon2domain; // new monitor
        m_current_phase[this] = mon2domain;
    end
    if(!m_current_phase[this].exists(dom)) begin
        phase_map_t domain2phase; // new domain
        m_current_phase[this][dom] = domain2phase;
    end
    if(!m_current_phase[this][dom].exists(imp)) begin
        m_current_phase[this][dom][imp] = phase; // rec'd
    end
endfunction
```

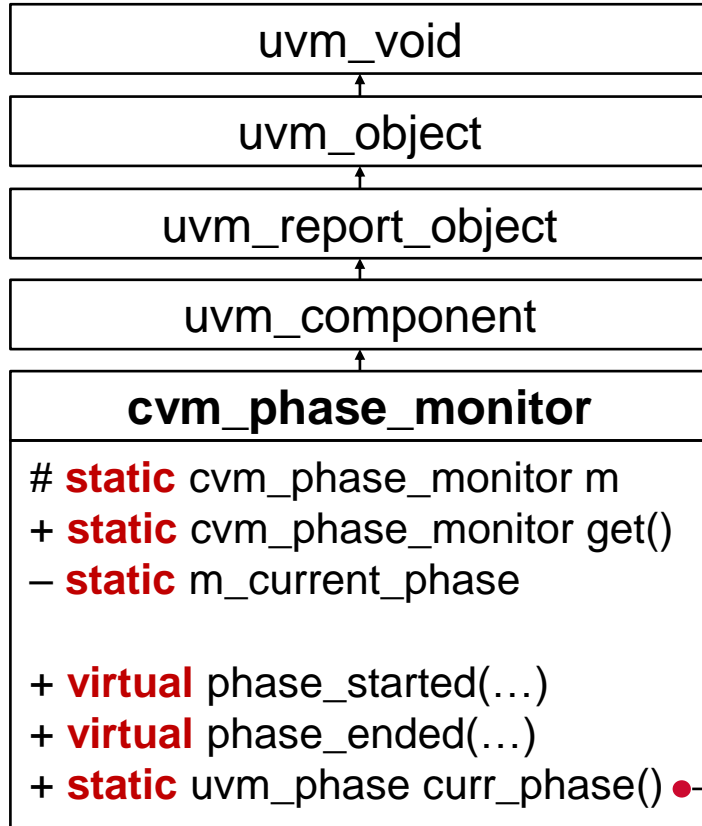
Phase Monitor



- **Delete** a phase from the array at end

```
function void phase_ended(uvm_phase phase);  
    uvm_domain dom = phase.get_domain();  
    uvm_phase imp = phase.get_imp();  
    if(!m_current_phase.exists(this)) begin end  
    else if(!m_current_phase[this].exists(dom)) begin end  
    else if(!m_current_phase[this][dom].exists(imp)) begin end  
    else begin  
        m_current_phase[this][dom].delete(imp); // delete phase  
        if(m_current_phase[this][dom].size() == 0) begin  
            m_current_phase[this].delete(dom); // delete empty domain  
            if(m_current_phase[this].size() == 0)  
                m_current_phase.delete(this); // delete empty array  
        end  
    end  
endfunction
```

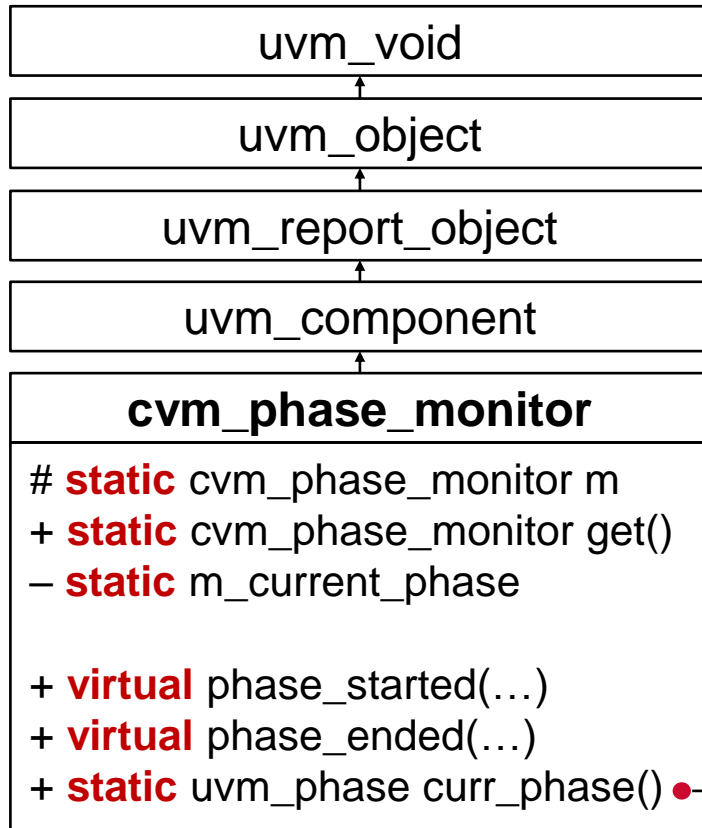
Phase Monitor



- **Retrieve** active phase from the array

```
function uvm_phase curr_phase();
    uvm_task_phase is_task;
    uvm_phase result = null;
    if(!m_current_phase.exists(m)) return null;
    foreach(m_current_phase[m][dom][imp])
    begin
        if($cast(is_task, imp)) // prefer task phase
            return m_current_phase[m][dom][imp];
        else if(result == null) // otherwise take first phase
            result = m_current_phase[m][dom][imp];
    end
    return result;
endfunction
```

Phase Monitor



- **Retrieve** active phase from

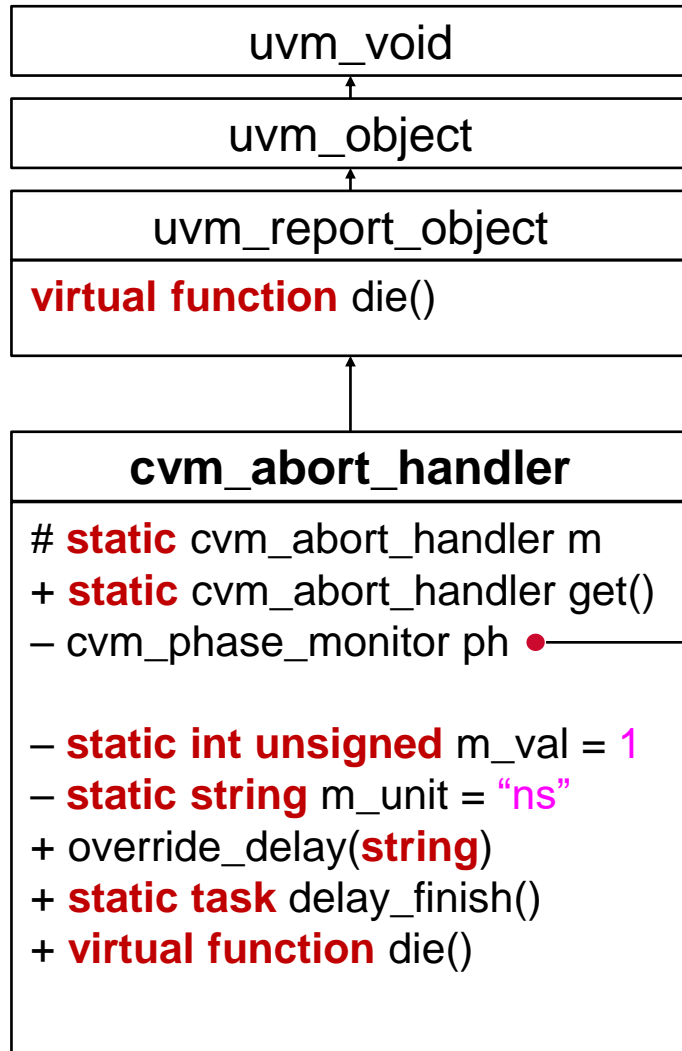
```
function uvm_phase curr_phase();
    uvm_task_phase is_task; uvm_phase result = null;
    if(m_current_phase.exists(m)) begin
        domain_map_t curr_d = m_current_phase[m];
        foreach(curr_d[dom]) begin
            phase_map_t curr_ph = curr_d[dom];
            foreach(curr_ph[imp]) begin
                if($cast(is_task, imp)) // prefer task phase
                    return curr_ph[imp];
                else if(result == null) // otherwise take first phase
                    result = curr_ph[imp];
            end
        end
    end
    return result;
endfunction
```

ALL
Compiler
Approved

Phasing Problem Defined

- During the delay to **\$finish** in TASK phases all objections may drop and sim exit
 - Raise a new objection at ``uvm_fatal`
 - Need to know active phase at ``uvm_fatal`
- During the delay to **\$finish** in FUNCTION phases objections are ignored
 - Do you want to introduce a delay?
 - Requires an external thread `post-run_test()`
 - Need to know active phase at ``uvm_fatal`

Custom Abort Handler with Phase Monitoring

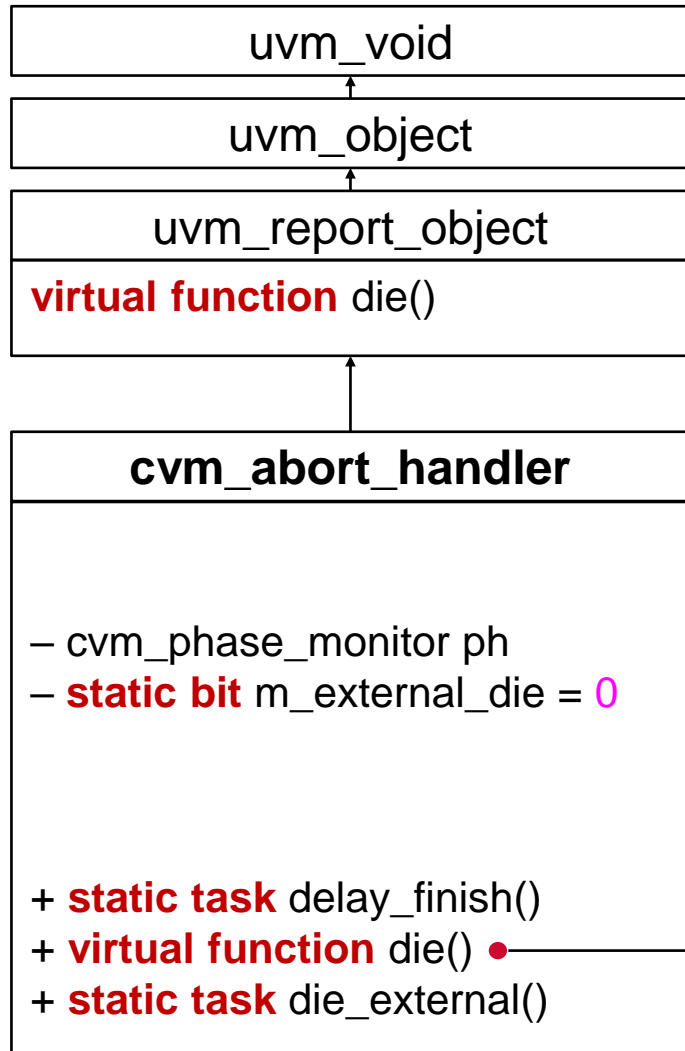


- Create the phase monitor singleton

```
`uvm_object_utils(cvm_abort_handler)
function void cvm_abort_handler get();
    if(m == null)
        m = cvm_abort_handler::type_id::create("abt", null);
endfunction

function new(string name = "cvm_abort_handler");
    super.new(name);
    ph = cvm_phase_monitor::get();
endfunction
```

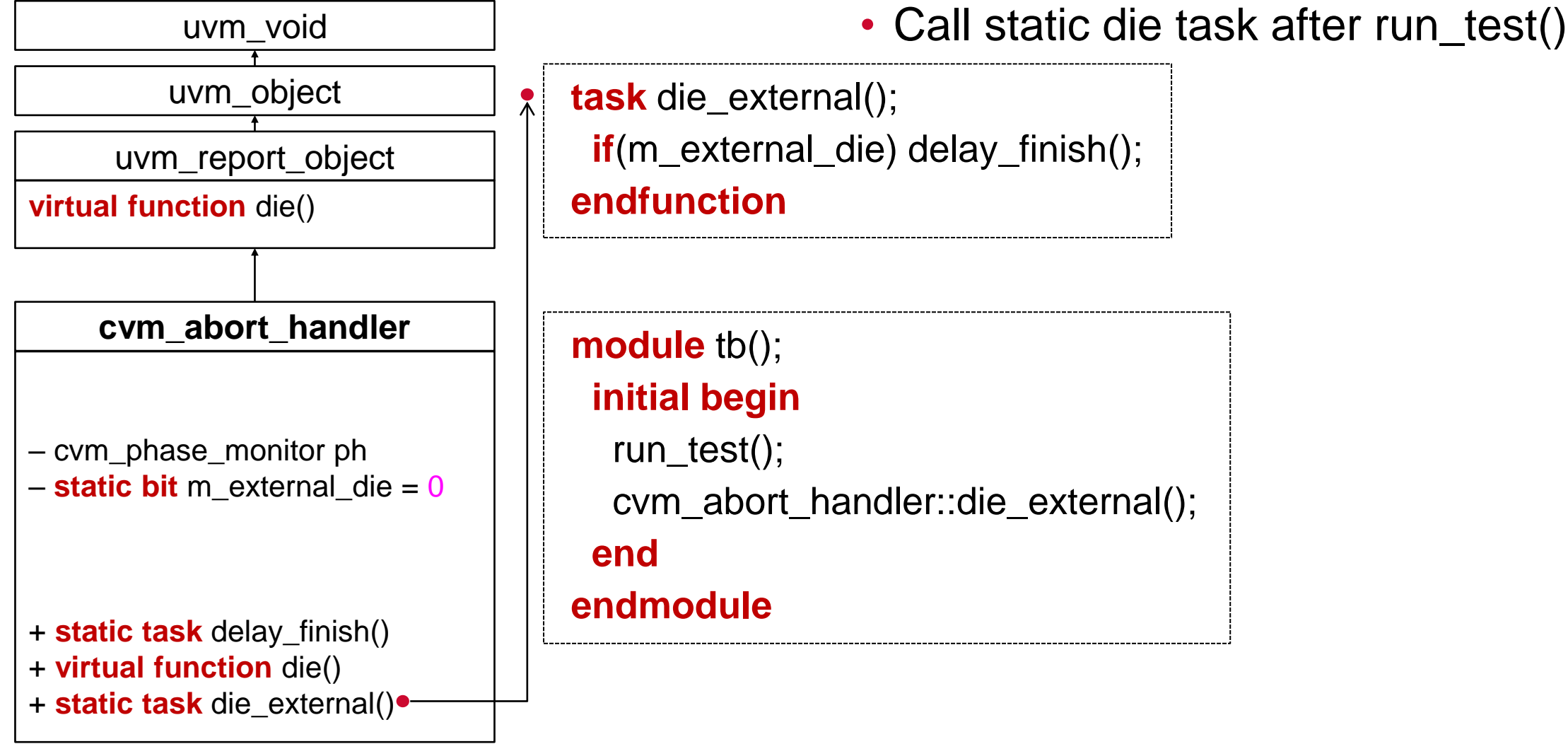

Custom Abort Handler with Phase Monitoring



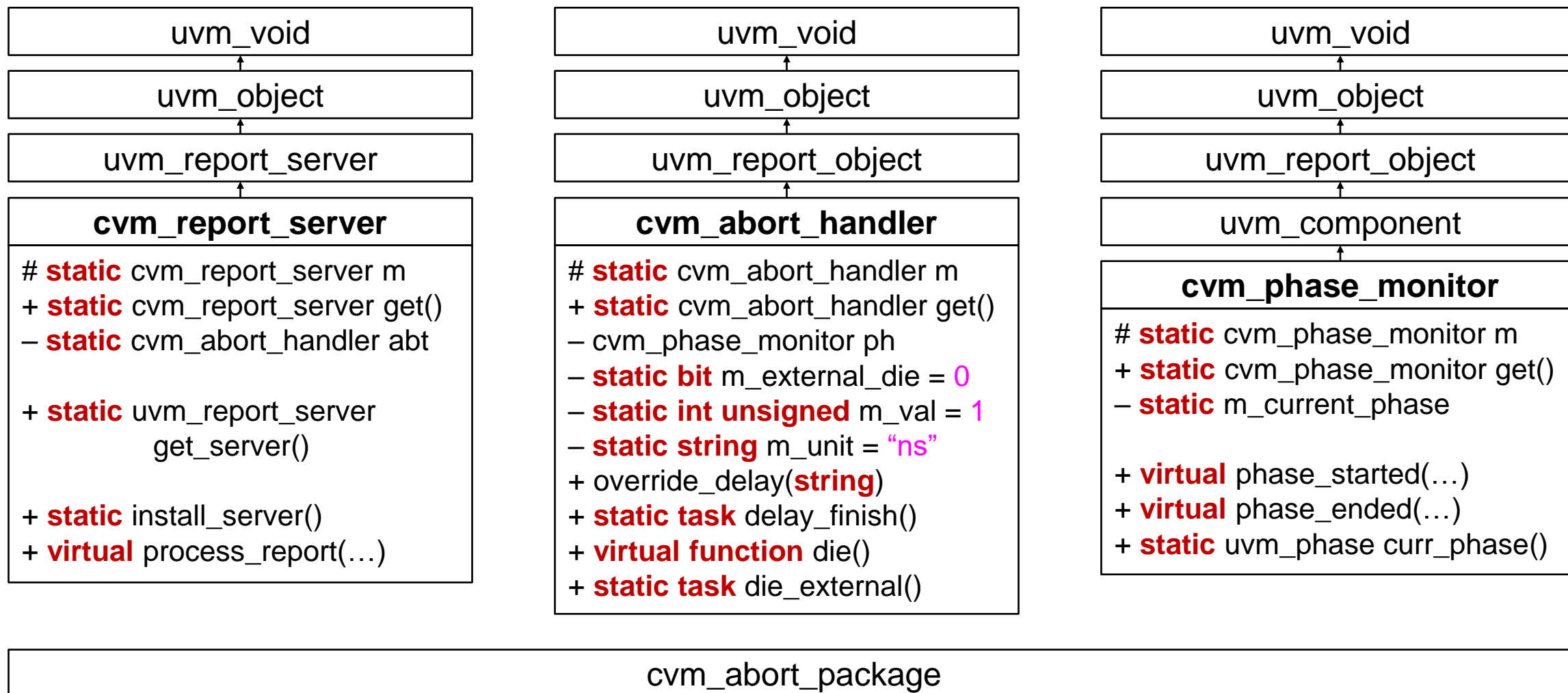
- Our die needs the phase and type

```
function void die();  
    uvm_root root = uvm_root::get();  
    uvm_phase phase = ph.curr_phase();  
    uvm_task_phase is_task;  
    root.m_do_pre_abort();  
    report_summarize();  
    if((phase != null) && ($cast(is_task, phase.get_imp())) begin  
        phase.raise_objection(root); // prevent phase exit  
        fork delay_finish(); join_none  
    end else begin  
        root.finish_on_completion = 0; // prevent run_test $finish call  
        m_external_die = 1; // enable post-run_test delay  
    end  
endfunction
```

Custom Abort Handler with Phase Monitoring



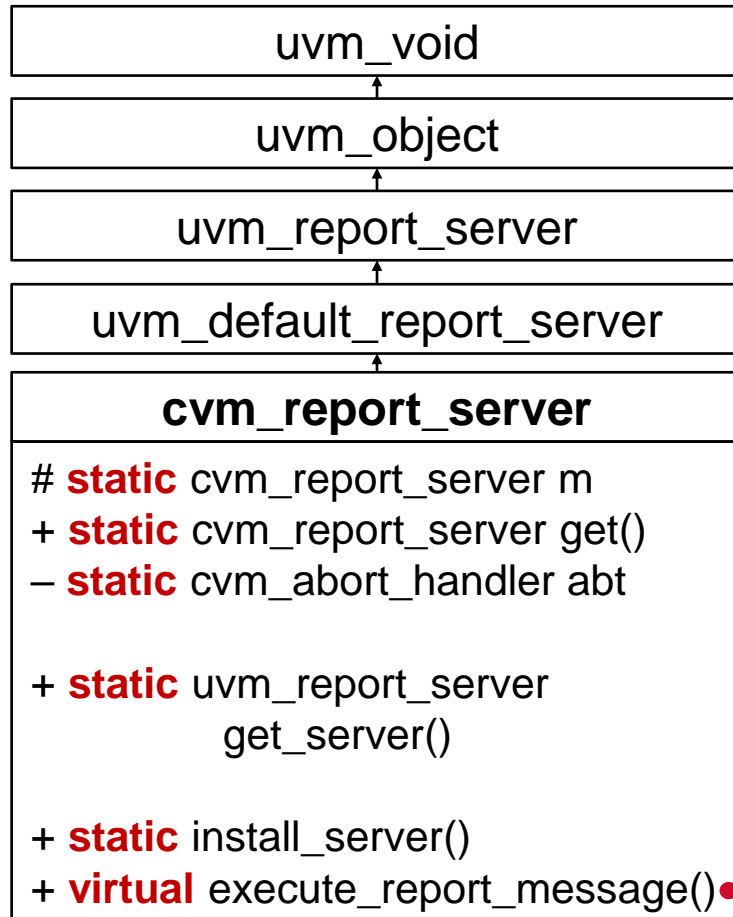
`uvm_fatal override architecture



Changes for UVM-1.2

- The die() function is now part of uvm_root
- Could “elegantly” extend:
 - uvm_root with updated die() function
 - uvm_coreservices_t with get_root() function returns extended root instance
 - Need to ``define UVM_CORESERVICE_TYPE` and somehow install new type into uvm_pkg?
 - Even if we achieve that ... does that mess up other things? The uvm_root is a global (package) space singleton constructed automatically: **const uvm_root** uvm_top = uvm_root::get();
- Or, just **hack the code**:
 - Extend uvm_default_report_server (no process_report() function exists)
 - Copy the whole execute_report_message() function and replace the necessary
- At your UVM library package install, copy from:
 - UVM_HOME/base/uvm_report_server.svh

Changes for UVM-1.2



- Wholesale override of function
 - do not call **super**.execute_report_message()

```
function void execute_report_message(
    uvm_report_message message, string composed_message);
... copy and paste ...

// Process the UVM_EXIT action
if(report_message.get_action() & UVM_EXIT) begin
    if(abt == null) begin // insert this
        uvm_root l_root;
        uvm_coreservice_t cs;
        cs = uvm_coreservice_t::get();
        l_root = cs.get_root();
        l_root.die();

        end else abt.die(); // and insert this
    ... copy and paste ...
endfunction
```

Automatic UVM Version Selection

```
package cvm_abort_package;  
  class cvm_phase_monitor extends uvm_component;  
endclass  
  class cvm_abort_handler extends uvm_report_object;  
endclass  
  class cvm_report_server  
    `ifdef UVM_VERSION_1_1  
      extends uvm_report_server;  
      virtual function void process_report(...); ... endfunction  
    `elsif UVM_VERSION_1_2  
      extends uvm_default_report_server;  
      virtual function void execute_report_message(...); ... endfunction  
    `endif  
  endclass  
endpackage : cvm_abort_package
```



Thank You

verificationhack.com





BROADCOM[®]

connecting everything[®]