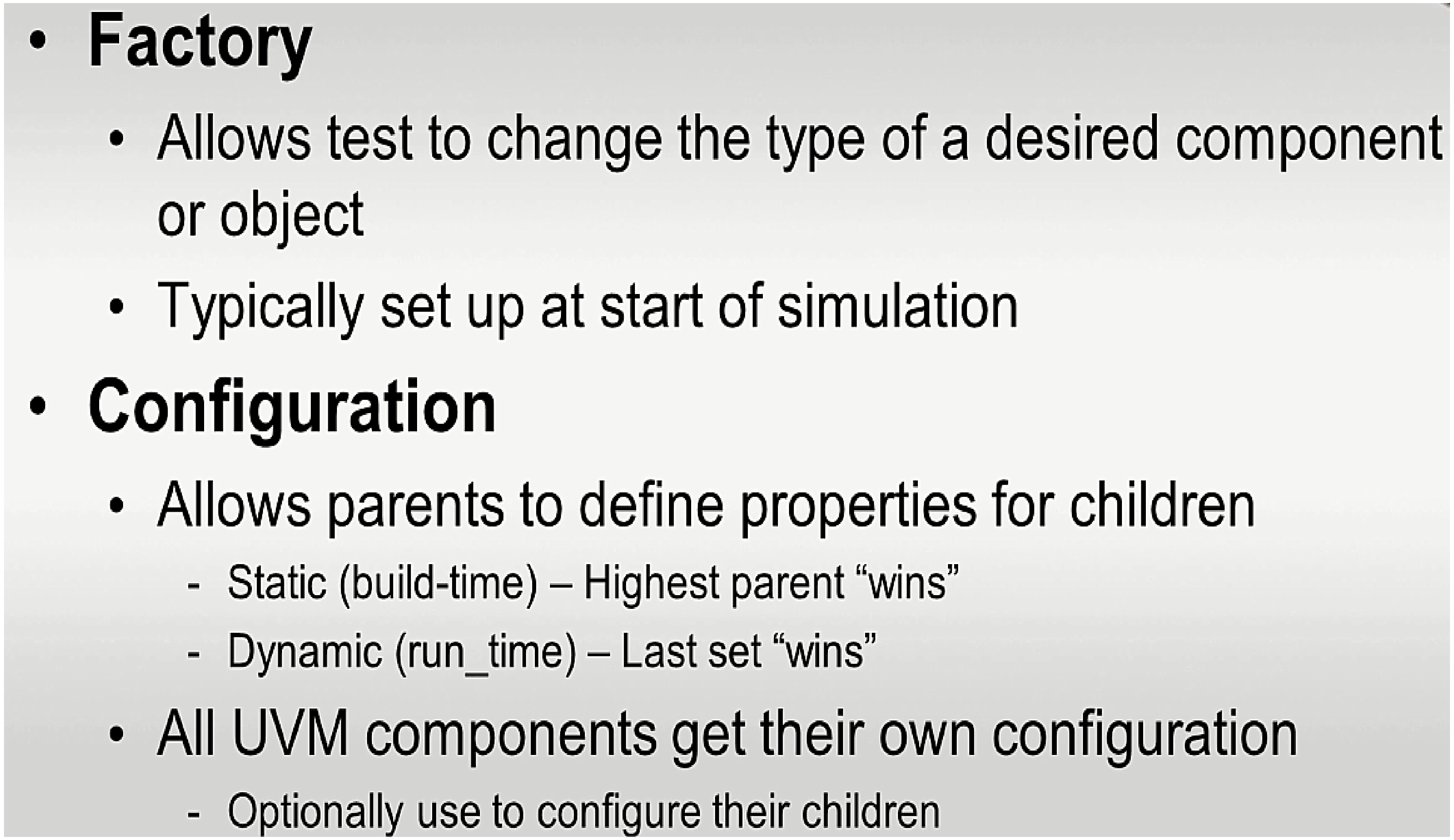
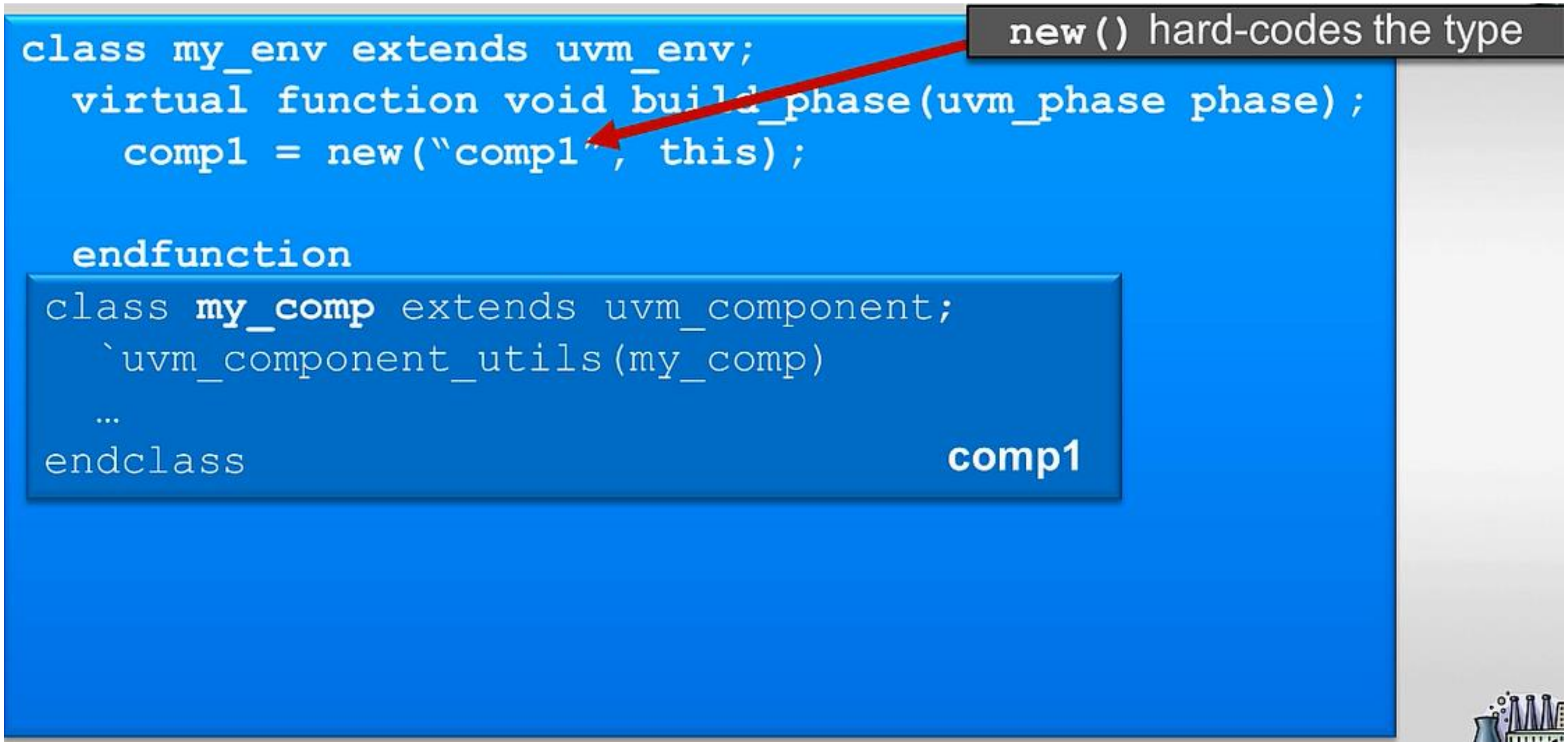
Two Customization Mechanisms



- **Factory**
  - Allows test to change the type of a desired component or object
  - Typically set up at start of simulation
- **Configuration**
  - Allows parents to define properties for children
    - Static (build-time) – Highest parent "wins"
    - Dynamic (run_time) – Last set "wins"
  - All UVM components get their own configuration
    - Optionally use to configure their children

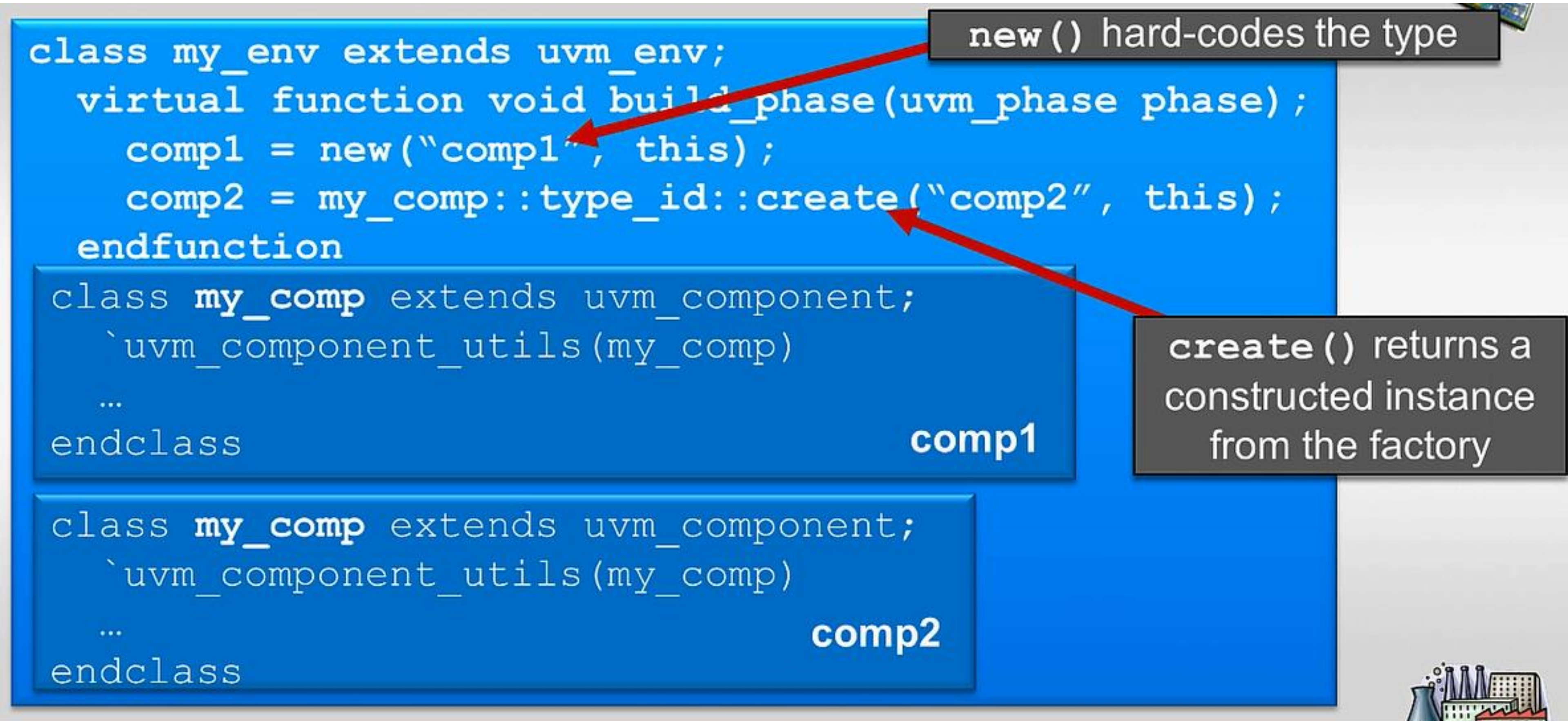
Create() vs new()



```
class my_env extends uvm_env;
  virtual function void build_phase(uvm_phase phase);
    comp1 = new("comp1", this);

  endfunction
```

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)
  …
endclass
```

Create() vs new()



```
class my_env extends uvm_env;
  virtual function void build_phase(uvm_phase phase);
    comp1 = new("comp1", this);
    comp2 = my_comp::type_id::create("comp2", this);
  endfunction
```

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)
  …
endclass
```

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)
  …
endclass
```

new() hard-codes the type
create() returns a constructed instance from the factory

Factory Methods

- Registration

```
`uvm_component_utils(class_type_name)
`uvm_component_param_utils(class_type_name #(params))
`uvm_object_utils(class_type_name)
`uvm_object_param_utils(class_type_name #(params))
```

- Examples
- **class** packet #(**type** T=**int**, **int** mode=0) **extends** uvm_object;

  `uvm_object_param_utils(packet #(T,mode))

  endclass
- **class** monitor #(**type** T=**int**, **int** mode=0) **extends** uvm_component;

  `uvm_component_param_utils(monitor #(T,mode))

  endclass

Factory Methods

- Construction **static function** T create(**string** name, uvm_component parent, **string context** = " ")

  - To construct a UVM based component or object, the static method create() should be used.
  - This function constructs the appropriate object based on the overrides, if any, and returns it.
  - The create() function returns an instance of the component type T, subject to any factory override based on the context provided by the parent’s full name.
  - The context argument, if supplied, supersedes the parents context.
  - The new instance will have the given leaf name and parent.

```
class_type object_name;
object_name = class_type::type_id::create("object_name",this);
```

Factory Methods

- Overriding
  - If required, user can override the registered classes or objects, based on name string or class-type.
  - For override by type, the override type is extended from original type.
  - The set_type_override() changes all instances of a class to a different class type.
  - The set_inst_override() changes specific instances of a class to a different class type.
  - For override by name, the original and override type names are class names that are registered in the factory.
  - When multiple overrides are done, by using the replace argument, we can control whether to override the previous override or not.
  - If replace is 1, then previous overrides will be replaced, otherwise, previous overrides will remain.

Factory Methods

- Overriding

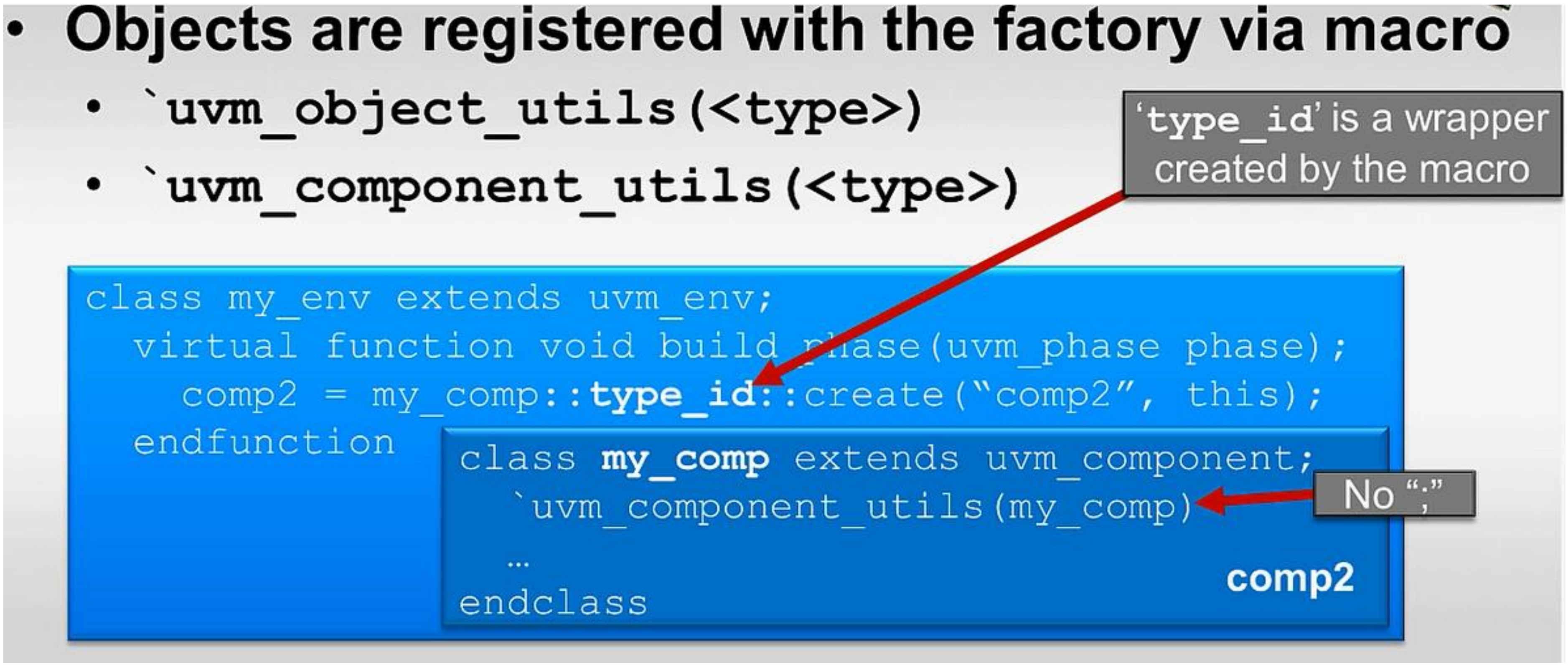
function void set_inst_override_by_type (uvm_object_wrapper original_type,
uvm_object_wrapper override_type,
string full_inst_path)

function void set_inst_override_by_name (**string** original_type_name,
string override_type_name,
string full_inst_path)

function void set_type_override_by_type (uvm_object_wrapper original_type,
uvm_object_wrapper override_type,
bit replace = 1)

function void set_type_override_by_name (**string** original_type_name,
string override_type_name,
bit replace = 1)

Registering with the factory



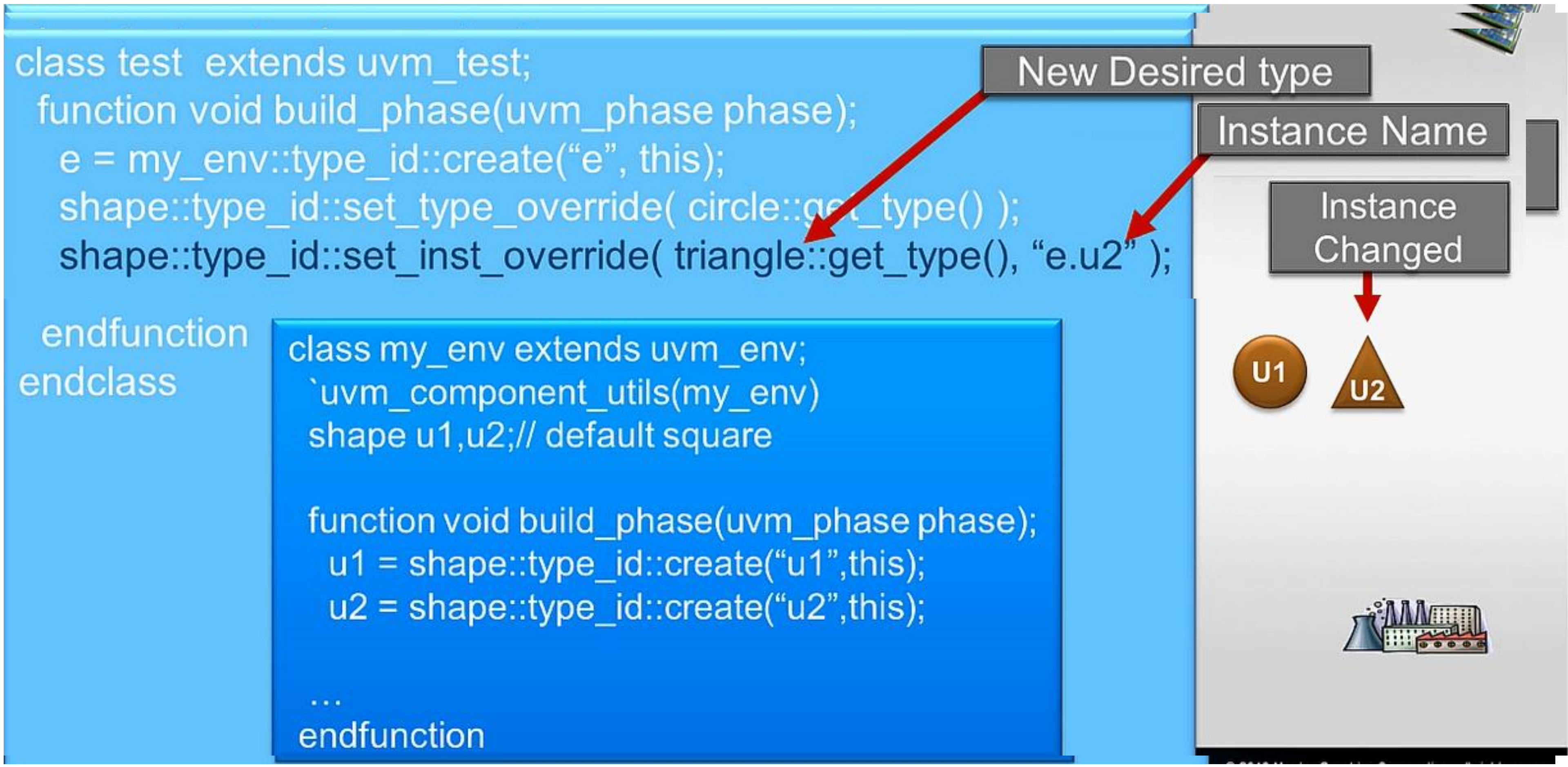
- **Objects are registered with the factory via macro**
  - **`` `uvm_object_utils(<type>) ``**
  - **`` `uvm_component_utils(<type>) ``**

```
class my_env extends uvm_env;
  virtual function void build_phase(uvm_phase phase);
    comp2 = my_comp::type_id::create("comp2", this);
  endfunction
```

```
class my_comp extends uvm_component;
  `uvm_component_utils(my_comp)
  …
endclass
```

'type_id' is a wrapper created by the macro
No ";"
comp2

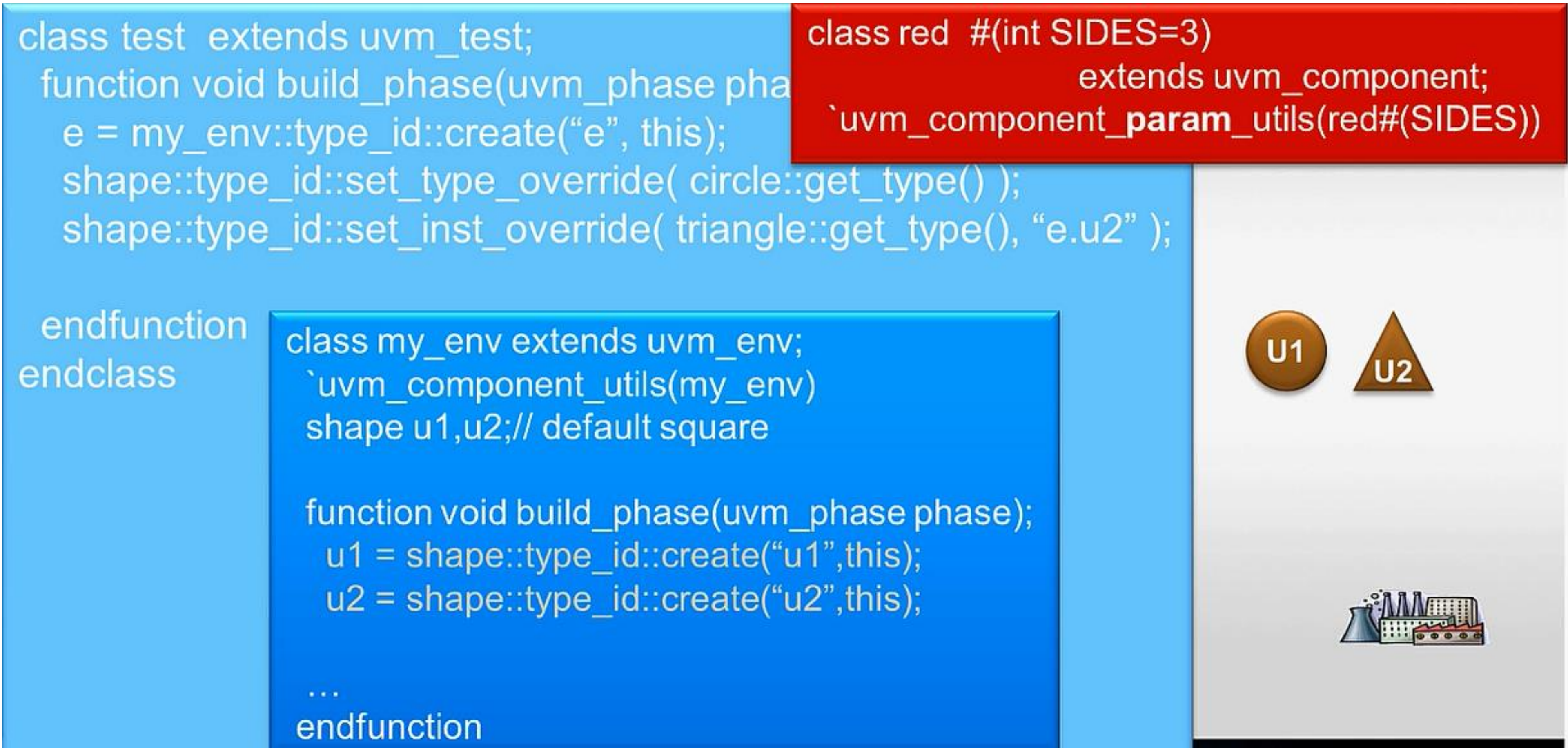
Overriding a type



```
class test  extends uvm_test;
 function void build_phase(uvm_phase phase);
   e = my_env::type_id::create("e", this);
   shape::type_id::set_type_override( circle::get_type() );
   shape::type_id::set_inst_override( triangle::get_type(), "e.u2" );

  endfunction
endclass
```

```
class my_env extends uvm_env;
 `uvm_component_utils(my_env)
 shape u1,u2;// default square

 function void build_phase(uvm_phase phase);
   u1 = shape::type_id::create("u1",this);
   u2 = shape::type_id::create("u2",this);


 ...
endfunction
```

Using Parameterized Types

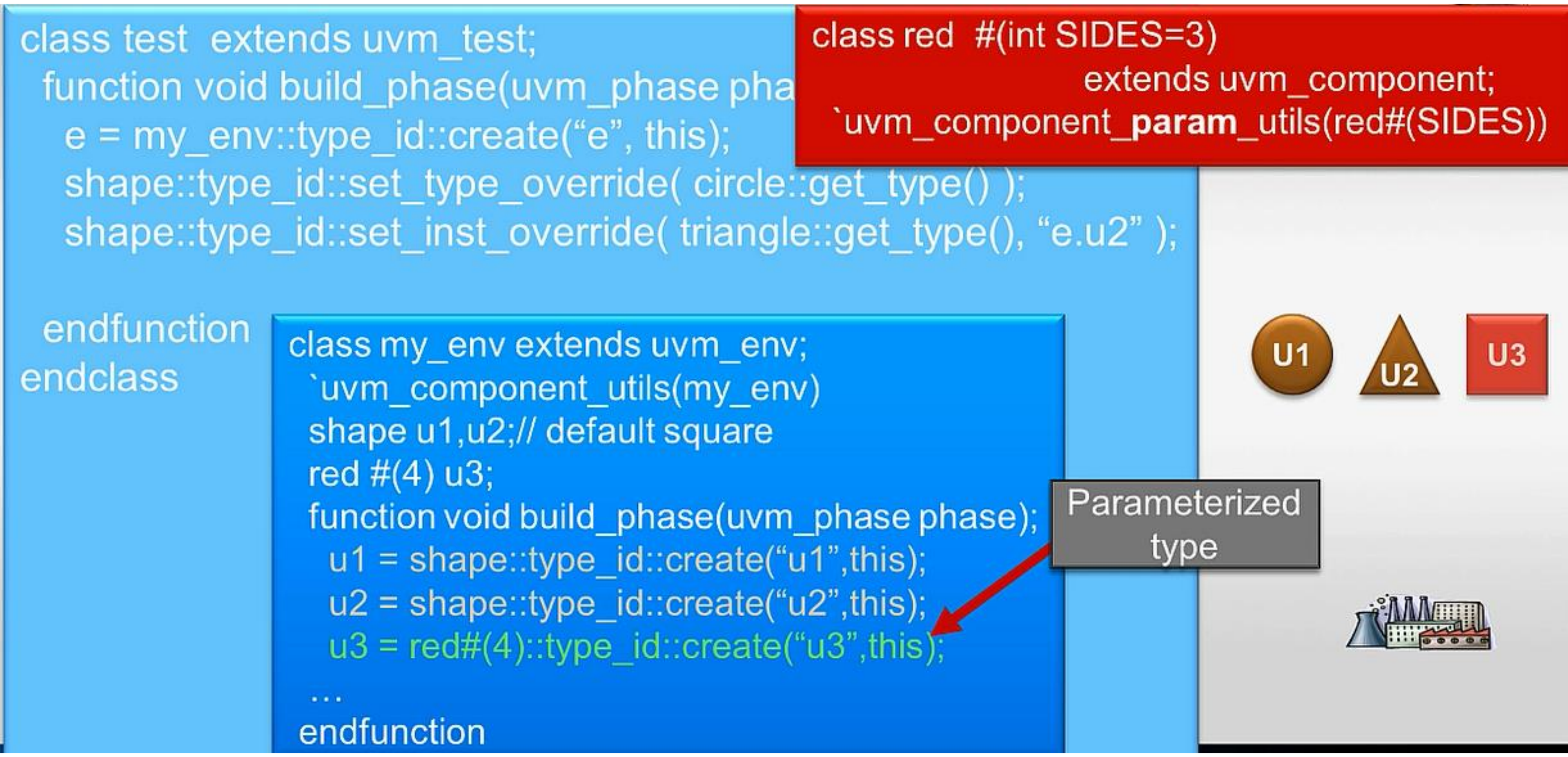


```
class test  extends uvm_test;
 function void build_phase(uvm_phase pha
  e = my_env::type_id::create("e", this);
  shape::type_id::set_type_override( circle::get_type() );
  shape::type_id::set_inst_override( triangle::get_type(), "e.u2" );

 endfunction
endclass
```

```
class red  #(int SIDES=3)
                       extends uvm_component;
 `uvm_component_param_utils(red#(SIDES))
```

```
class my_env extends uvm_env;
 `uvm_component_utils(my_env)
 shape u1,u2;// default square

 function void build_phase(uvm_phase phase);
  u1 = shape::type_id::create("u1",this);
  u2 = shape::type_id::create("u2",this);

 …
endfunction
```

Using Parameterized Types

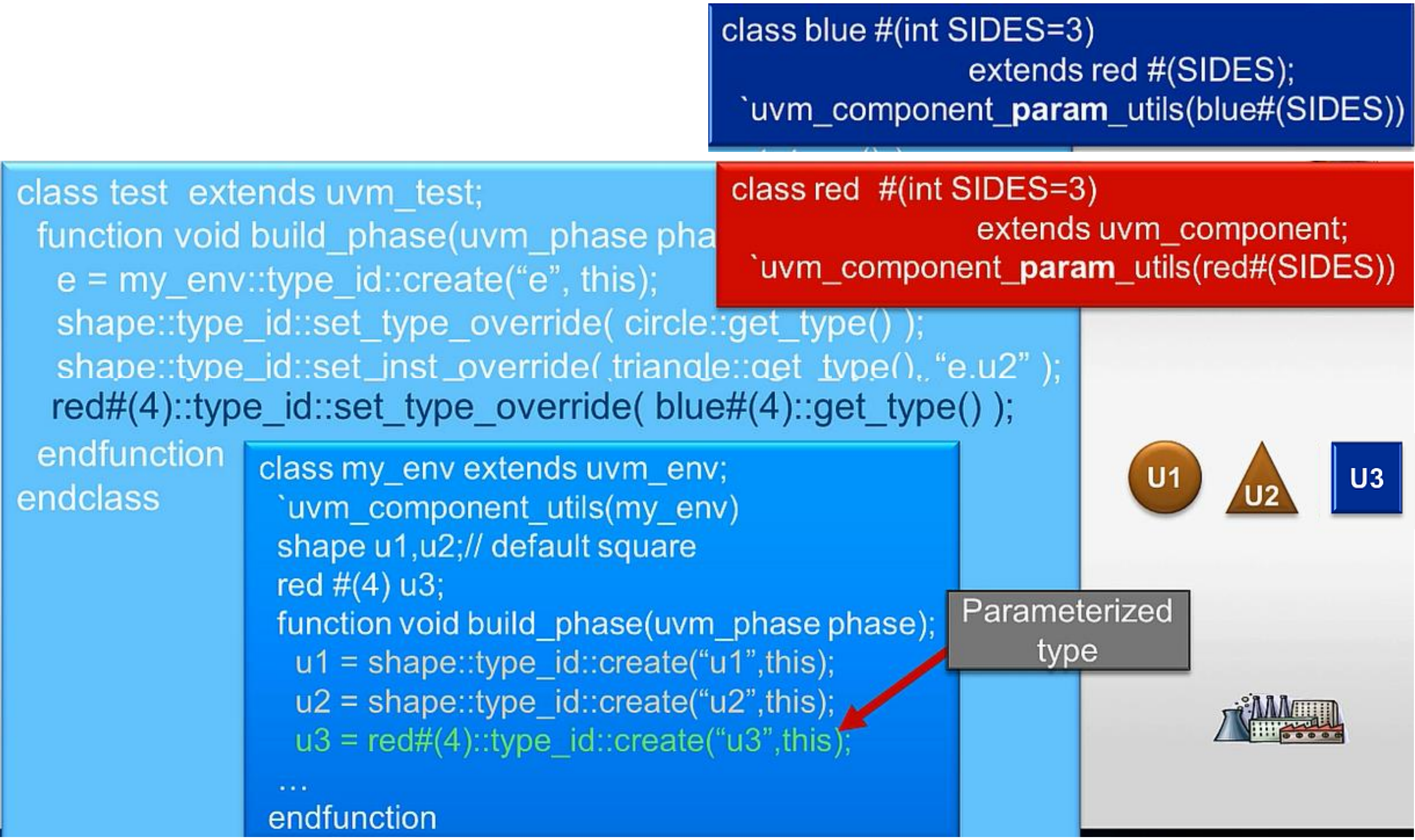


```
class test  extends uvm_test;
  function void build_phase(uvm_phase pha
    e = my_env::type_id::create("e", this);
    shape::type_id::set_type_override( circle::get_type() );
    shape::type_id::set_inst_override( triangle::get_type(), "e.u2" );

  endfunction
endclass
```

```
class red  #(int SIDES=3)
                extends uvm_component;
  `uvm_component_param_utils(red#(SIDES))
```

```
class my_env extends uvm_env;
  `uvm_component_utils(my_env)
  shape u1,u2;// default square
  red #(4) u3;
  function void build_phase(uvm_phase phase);
    u1 = shape::type_id::create("u1",this);
    u2 = shape::type_id::create("u2",this);
    u3 = red#(4)::type_id::create("u3",this);
  ...
 endfunction
```

Using Parameterized Types



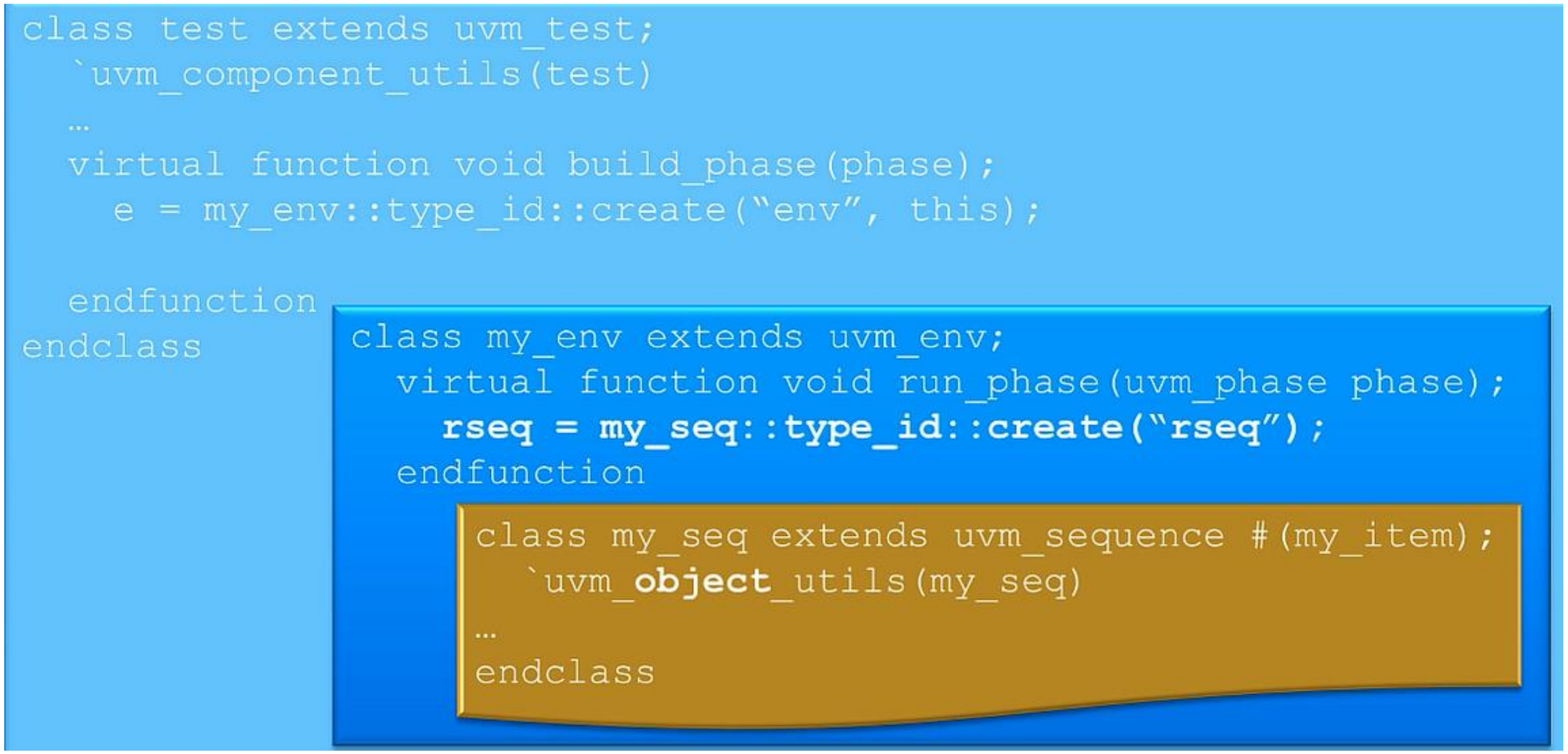
```
class blue #(int SIDES=3)
                    extends red #(SIDES);
  `uvm_component_param_utils(blue#(SIDES))
```

```
class red  #(int SIDES=3)
                    extends uvm_component;
  `uvm_component_param_utils(red#(SIDES))
```

```
class test  extends uvm_test;
  function void build_phase(uvm_phase pha
    e = my_env::type_id::create("e", this);
    shape::type_id::set_type_override( circle::get_type() );
    shape::type_id::set_inst_override( triangle::get_type(), "e.u2" );
   red#(4)::type_id::set_type_override( blue#(4)::get_type() );
  endfunction
endclass
```

```
class my_env extends uvm_env;
  `uvm_component_utils(my_env)
  shape u1,u2;// default square
  red #(4) u3;
  function void build_phase(uvm_phase phase);
    u1 = shape::type_id::create("u1",this);
    u2 = shape::type_id::create("u2",this);
    u3 = red#(4)::type_id::create("u3",this);
  …
 endfunction
```

Use the factory for objects too

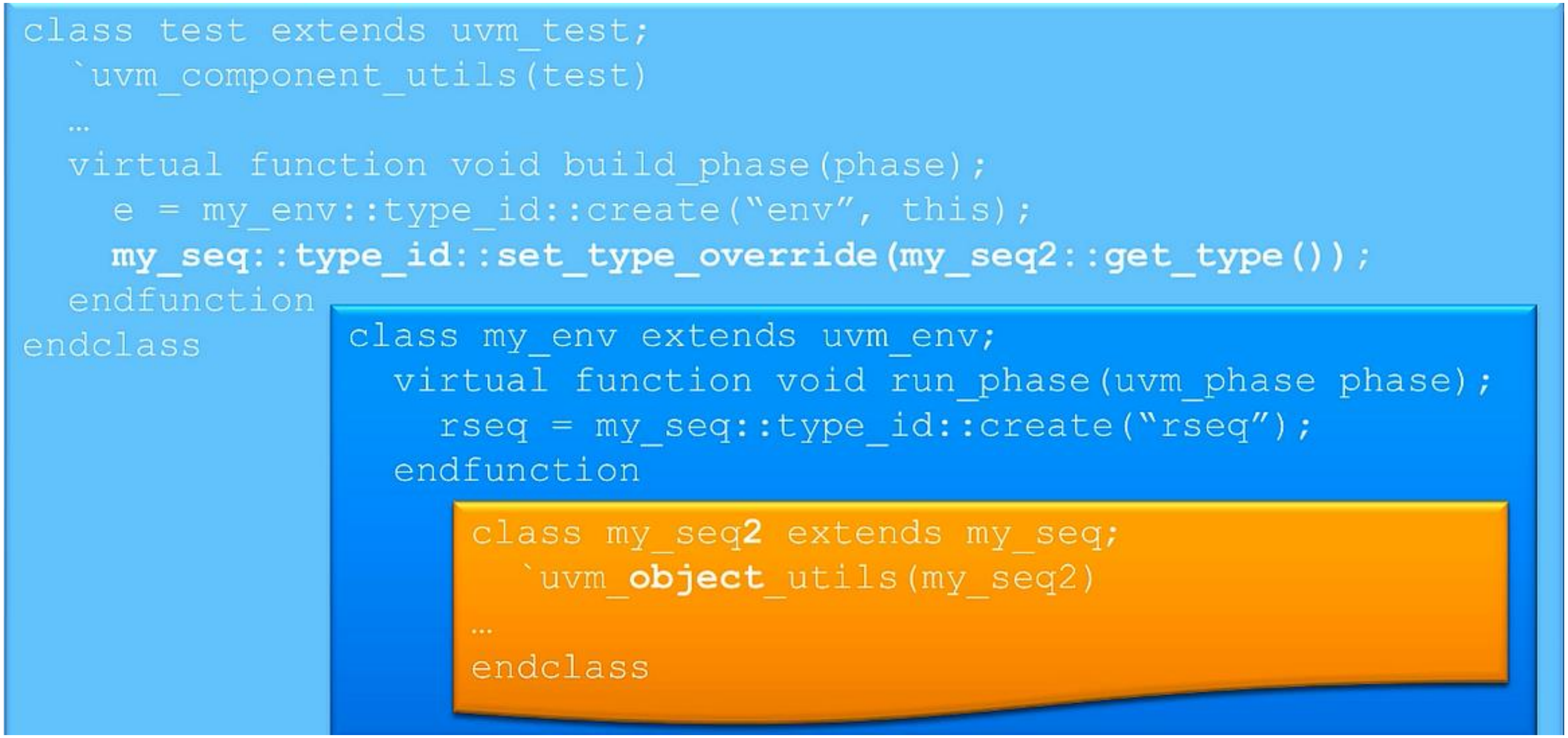


```
class test extends uvm_test;
  `uvm_component_utils(test)
  …
  virtual function void build_phase(phase);
    e = my_env::type_id::create("env", this);

  endfunction
endclass
```

```
class my_env extends uvm_env;
  virtual function void run_phase(uvm_phase phase);
    rseq = my_seq::type_id::create("rseq");
  endfunction
```

```
class my_seq extends uvm_sequence #(my_item);
  `uvm_object_utils(my_seq)
…
endclass
```

Use the factory for objects too

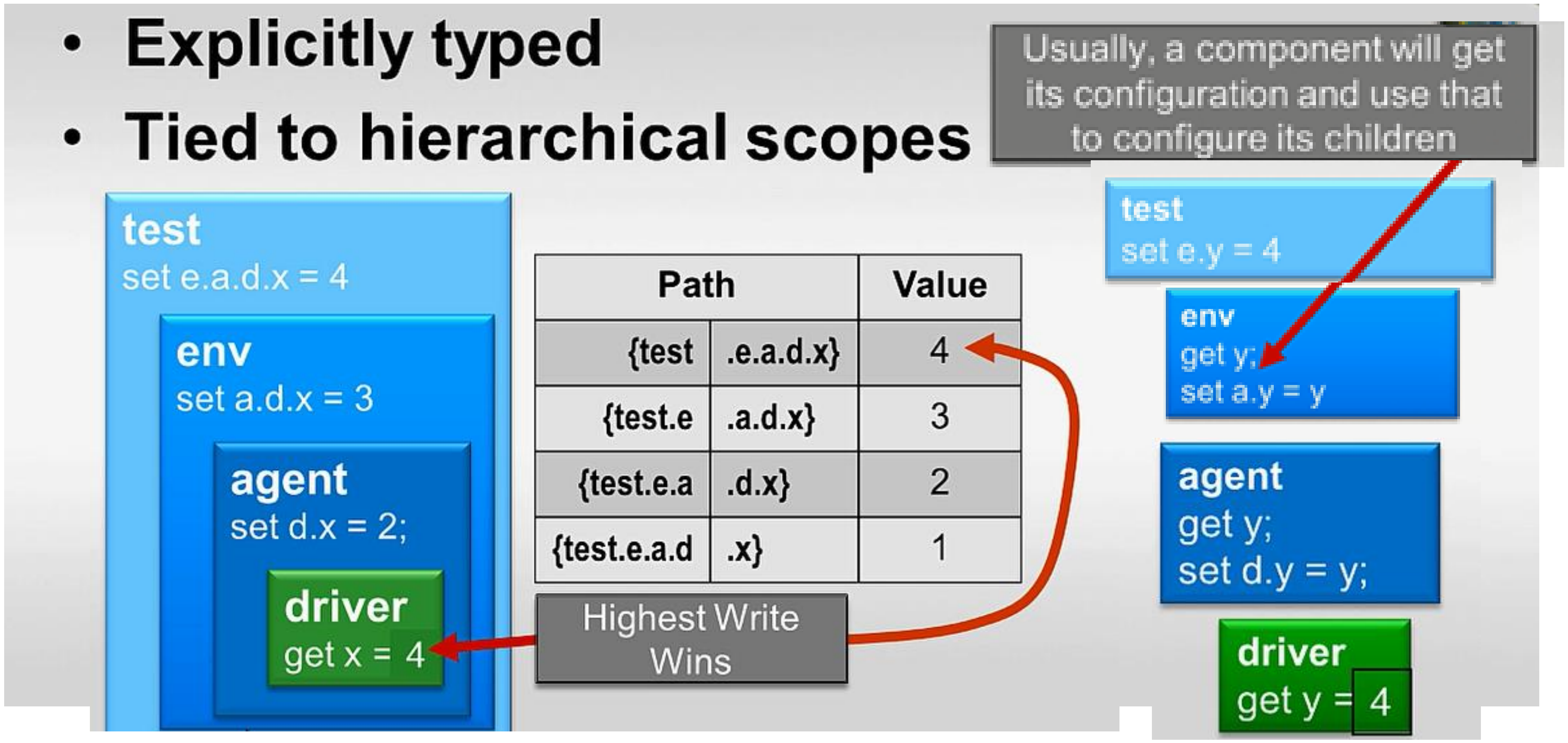


```
class test extends uvm_test;
  `uvm_component_utils(test)
  …
  virtual function void build_phase(phase);
    e = my_env::type_id::create("env", this);
    my_seq::type_id::set_type_override(my_seq2::get_type());
  endfunction
endclass
```

```
class my_env extends uvm_env;
  virtual function void run_phase(uvm_phase phase);
    rseq = my_seq::type_id::create("rseq");
  endfunction
```

```
class my_seq2 extends my_seq;
  `uvm_object_utils(my_seq2)
…
endclass
```

UVM Configuration Database



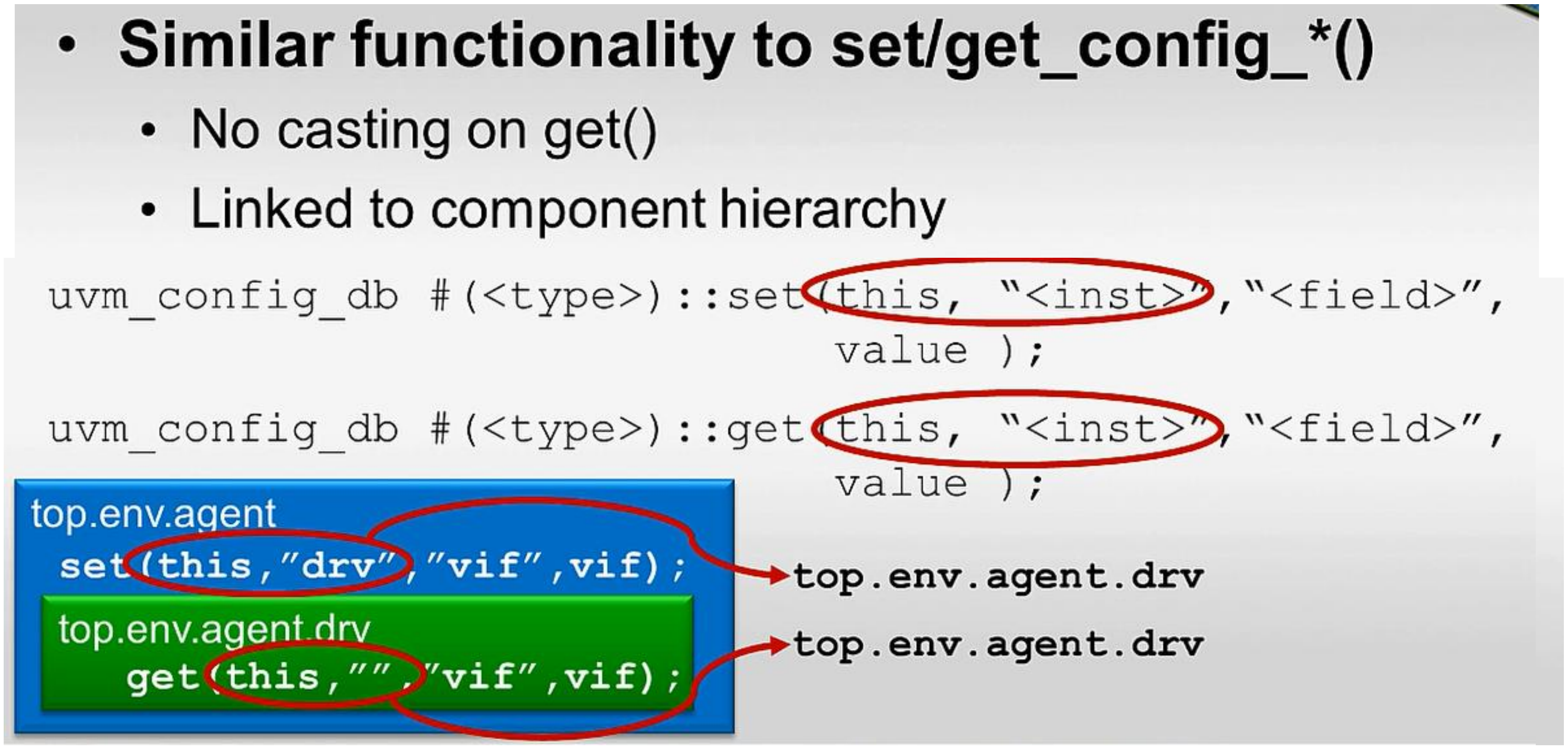
- **Explicitly typed**
- **Tied to hierarchical scopes**

Path		Value
{test	.e.a.d.x}	4
{test.e	.a.d.x}	3
{test.e.a	.d.x}	2
{test.e.a.d	.x}	1

Highest Write Wins

Usually, a component will get its configuration and use that to configure its children

uvm_config_db

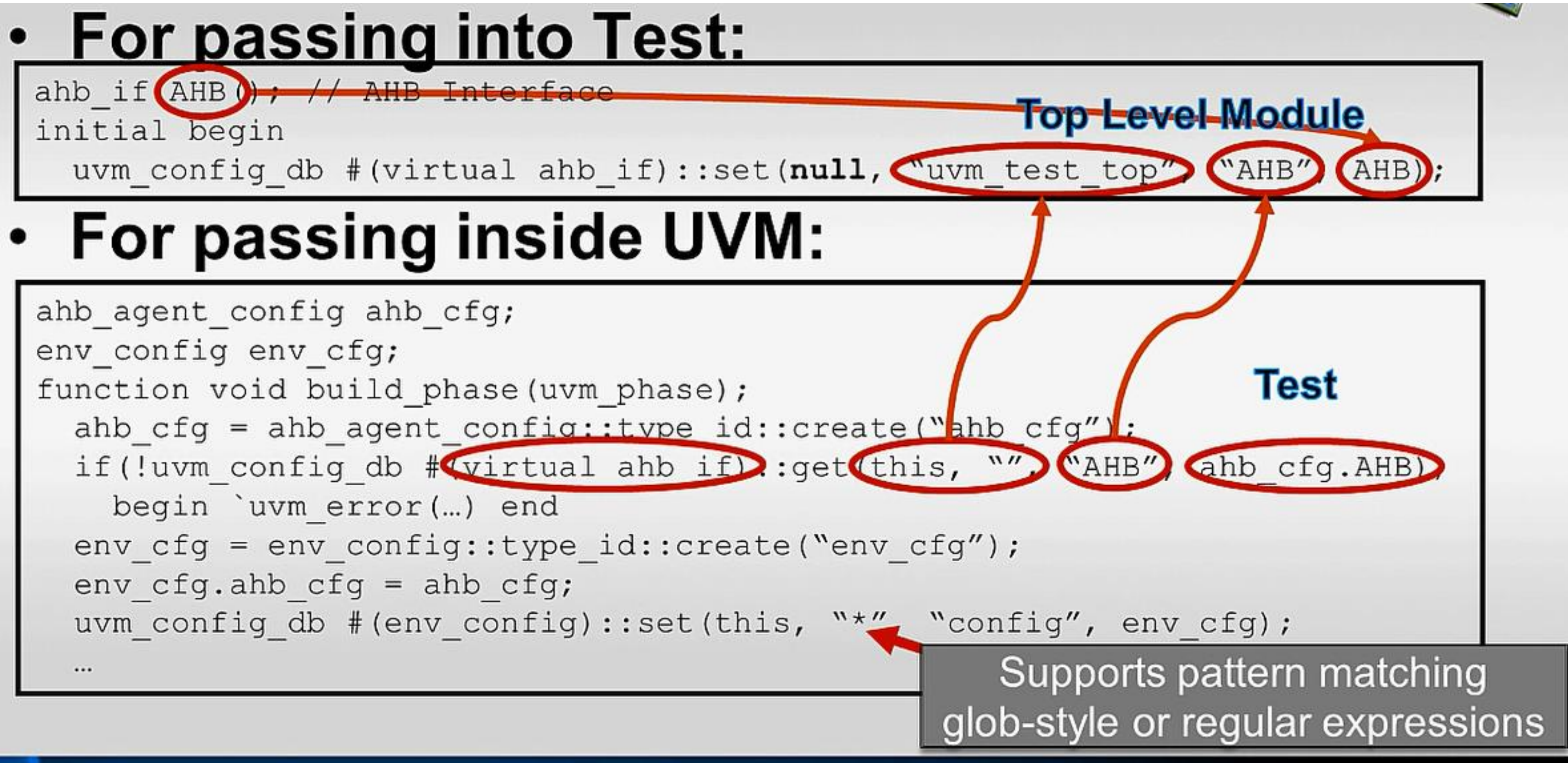


- **Similar functionality to set/get_config_*()**
  - No casting on get()
  - Linked to component hierarchy

```
uvm_config_db #(<type>)::set(this, "<inst>","<field>",
                             value );
uvm_config_db #(<type>)::get(this, "<inst>","<field>",
                             value );
```

top.env.agent
set(this,"drv","vif",vif);
top.env.agent.drv
get(this,"","vif",vif);
top.env.agent.drv
top.env.agent.drv

UVM Features-uvm_config_db



- **For passing into Test:**

```
ahb_if AHB(); // AHB Interface
initial begin
  uvm_config_db #(virtual ahb_if)::set(null, "uvm_test_top", "AHB", AHB);
```

- **For passing inside UVM:**

```
ahb_agent_config ahb_cfg;
env_config env_cfg;
function void build_phase(uvm_phase);
  ahb_cfg = ahb_agent_config::type_id::create("ahb_cfg");
  if(!uvm_config_db #(virtual ahb_if)::get(this, "", "AHB", ahb_cfg.AHB))
    begin `uvm_error(…) end
  env_cfg = env_config::type_id::create("env_cfg");
  env_cfg.ahb_cfg = ahb_cfg;
  uvm_config_db #(env_config)::set(this, "*", "config", env_cfg);
  …
```