2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

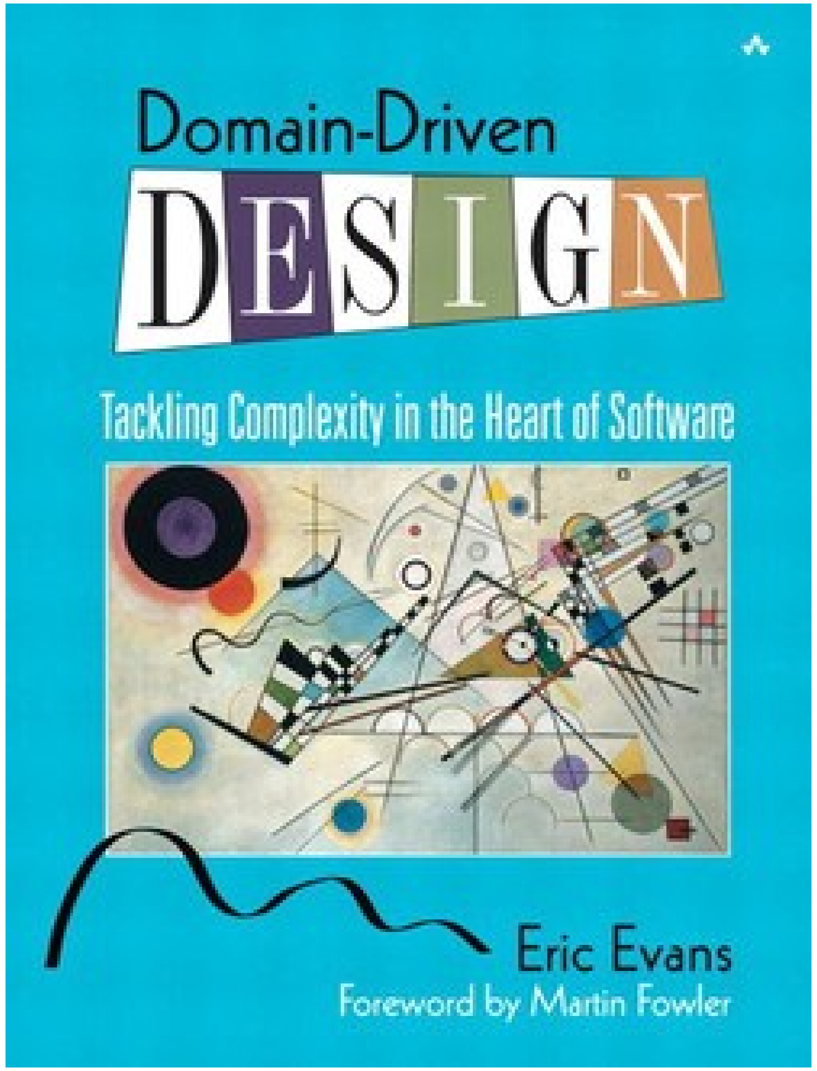
Don’t Go Changing: How to Code Immutable UVM Objects

William L. Moore

accellera
SYSTEMS INITIATIVE

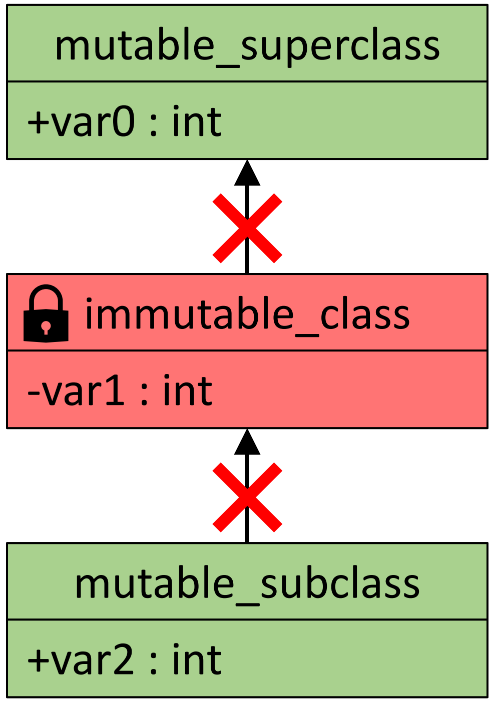
Immutable Value Objects

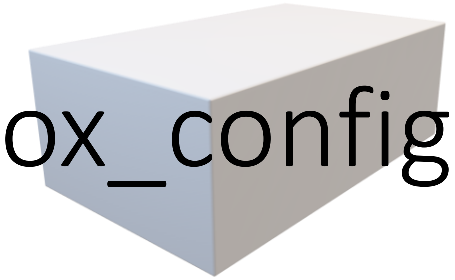
- Object-oriented programming (OOP)
- Domain-driven design (DDD)
- Value objects are defined by values
- No identity, *cf. entities*
- Immutable objects initialized at creation
- Constant: values never change
- Spectrum of immutability


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

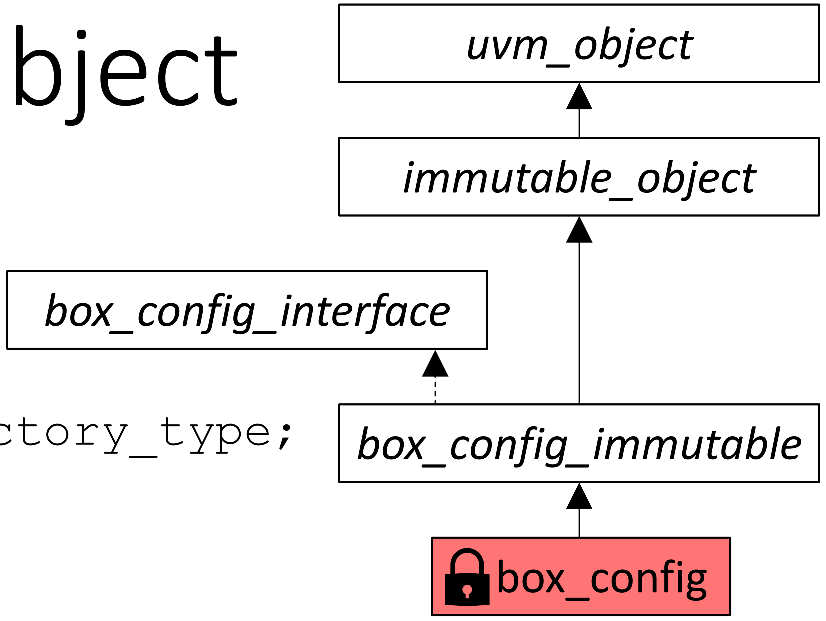
How to Code Immutable SystemVerilog Objects

- Local variables with public "getter" functions
- No public or protected "setters"
- Initialize values through constructor parameters
- Don't go changing values
- Don't share references to mutable objects
- No random variables
- Final—no subclasses or mutable superclasses





box_config: Immutable UVM Object



```
class box_config extends box_config_immutable;

typedef box_config_factory_generic#(box_config) factory_type;
local int length, width, height;
`uvm_field_utils_begin(box_config)
`uvm_field_int(length, UVM_ALL_ON | UVM_NOPACK | UVM_NOCOPY | UVM_READONLY);
`uvm_field_int(width , UVM_ALL_ON | UVM_NOPACK | UVM_NOCOPY | UVM_READONLY);
`uvm_field_int(height, UVM_ALL_ON | UVM_NOPACK | UVM_NOCOPY | UVM_READONLY);
`uvm_field_utils_end
```

box_config (cont.): Constructor

```
local function new (string name="", int length=0, int width=0, int height=0);
    super.new(name);
    this.set_length(length);
    this.set_width(width);
    this.set_height(height);
endfunction
```

box_config (cont.): Static Factory Methods

```
static function box_config_immutable create_new (
        string name="", int length=0, int width=0, int height=0 );
    box_config product = new(name, length, width, height);
    return product;
endfunction

static function box_config_immutable create_copy (
        string name="", uvm_object rhs);
    create_copy = box_config_copier#(box_config)::create_copy(name, rhs);
endfunction
```

box_config (cont.): Required uvm_object Methods

```
virtual function string get_type_name ();
    return "box_config";
endfunction

virtual function uvm_object create (string name="");
    box_config object = new(name);
    return object;
endfunction
```

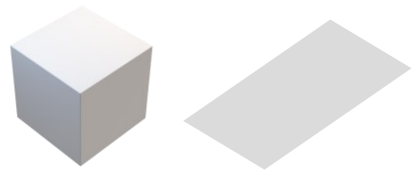
box_config (cont.): Public Getters, Local Setters

```
virtual function int get_length ();
    return this.length;
endfunction

local function void set_length (int length);
    this.length = length;
endfunction
...
endclass
```

But What About the UVM Factory?!

- No `` `uvm_object_utils `` registration, no UVM factory!
- No UVM factory, no overrides
- Polymorphic family of `box_config_immutable` variants
- Solution: secondary registered factory creates immutables



```
box_config_factory factory;
factory = box_config_factory::type_id::create("factory");
box_cfg = factory.create_new("box_cfg", length, width, height);
```

- Overriding secondary factory produces different immutables

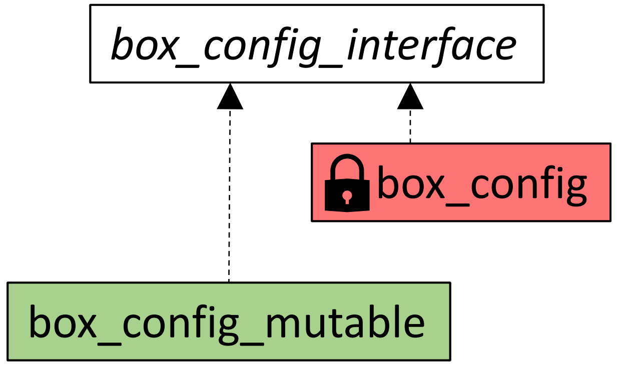
Parameterized Factory Produces Whole Family

```
class box_config_factory_generic#(type PT=box_config) extends box_config_factory;
    `uvm_object_param_utils(box_config_factory_generic#(PT))
    function new (string name="box_config_factory_generic");
        super.new(name);
    endfunction

    virtual function box_config_immutable create_new (
            string name="", int length=0, int width=0, int height=0);
        create_new = PT::create_new(name, length, width, height);
    endfunction

    virtual function box_config_immutable create_copy (
            string name="", uvm_object rhs);
        create_copy = PT::create_copy(name, rhs);
    endfunction
endclass
```

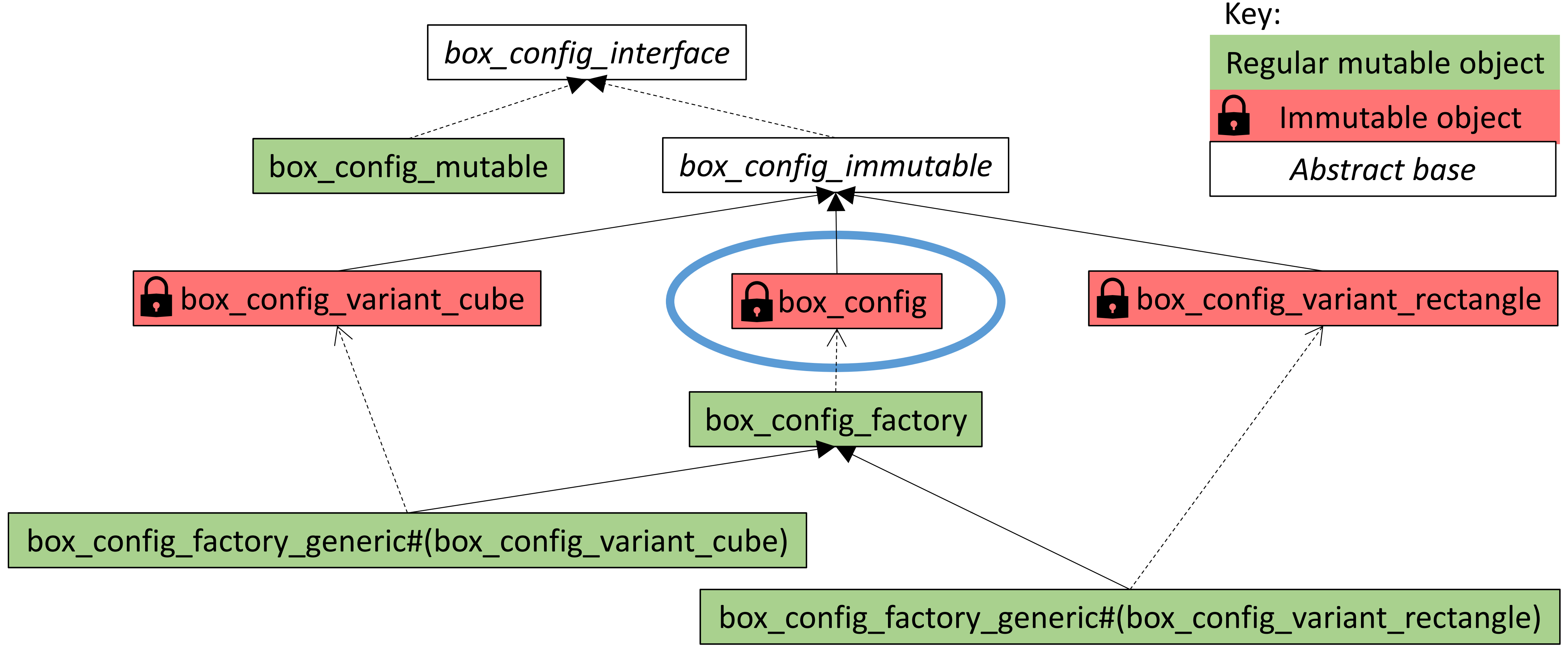
But What About Randomization?!



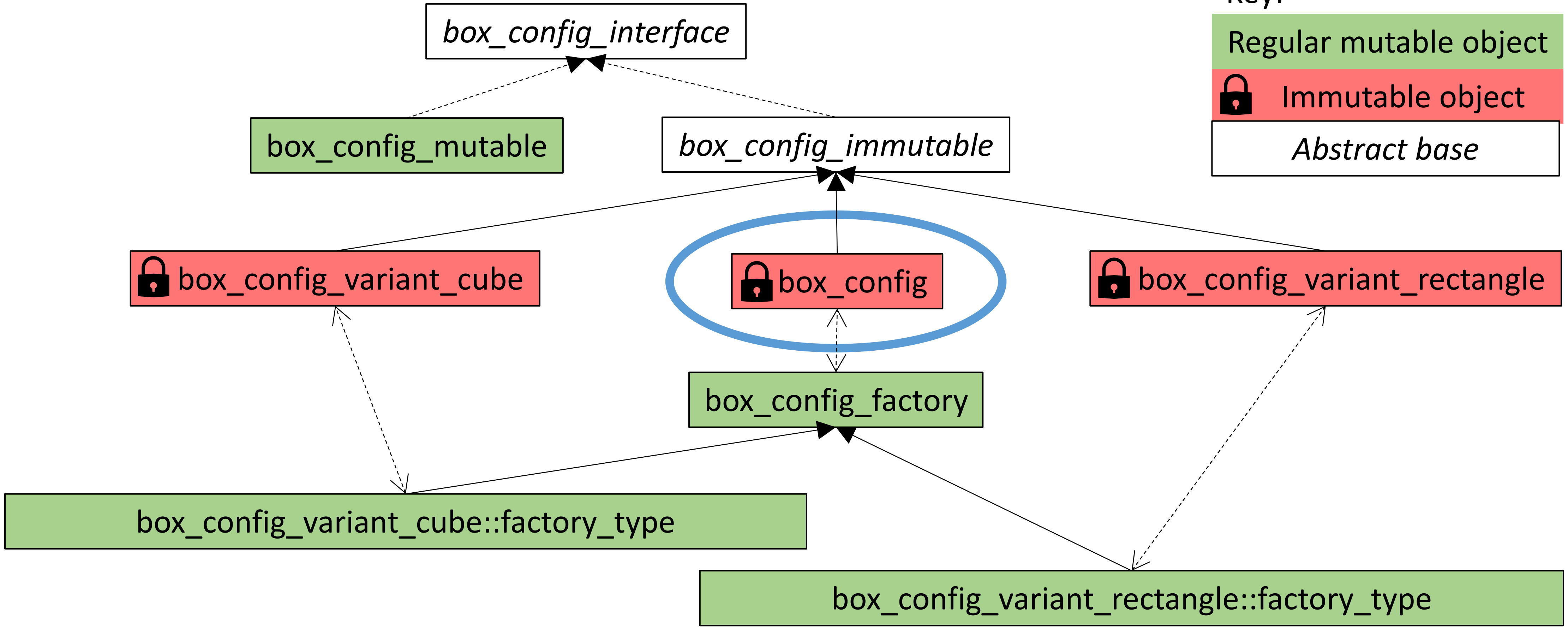
- Randomization is mutation, which is forbidden!
- Solution: mutable version of immutable class
- Public constrainable `rand` variables
- Shared base class for polymorphic compatibility
- Copy constructors allow two-way conversion

```
box_config_mutable temp = box_config_mutable::type_id::create("temp");
void'(temp.randomize());
box_cfg = box_config::create_copy("box_cfg", temp);
```

Meet the Family



Meet the Family



Half-baked Alternative Constructor Knob Strategies

- Intermediary, e.g., `uvm_resource_db`, global variables
- Static class variables
- Formatted `name` parameter string (`$sformatf`/`$sscanf`)
- Non-printable `name` string packed with `uvm_packer`
- Class parameters
- `uvm_component` constructor `parent` component parameter

```
function new (string name, uvm_component parent);
```

H.A.C.K.S.

- Intermediary, e.g., `uvm_resource_db`, global variables
- Static class variables
- Formatted `name` parameter string (`$sformatf`/`$sscanf`)
- Non-printable `name` string packed with `uvm_packer`
- Class parameters
- `uvm_component` constructor `parent` component parameter

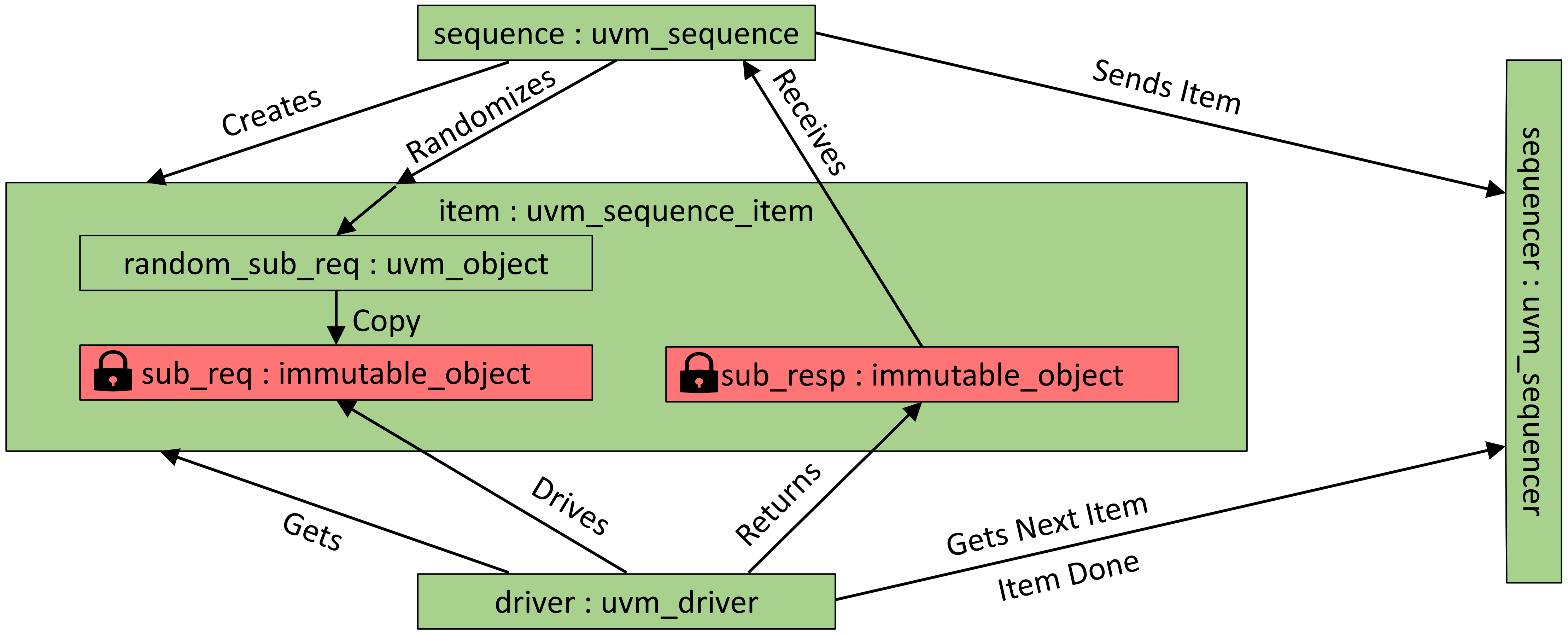
```
function new (string name, uvm_component parent);
```

Composing a Sequence Item

- `uvm_sequence_item` models transactions
- Not value object, changes over time
- Treated like a mutable value object
- Consider composing with sub-sequence items
  - Mutable value objects for random stimulus
  - Immutable copies of stimulus for sharing
  - Immutable snapshots of observed values

```
class simple_trans extends
        uvm_sequence_item;
    rand data_t data;
    rand addr_t addr;
    rand enum {WRITE,READ} kind;
    ...
```

Sequence Item Flow



Conclusion

- Drawbacks
  - Developer time and effort
  - Scalability and maintenance challenges
  - Difficult to rework legacy code
  - Extra steps for user

- Benefits
  - Modularity, model fidelity
  - Class cohesion, separation of concerns
  - Reuse, unit testability, fewer defects
  - Clarity of ownership and relationships
  - Sharing without aliasing bugs, hazards, corruption
  - Simpler interfaces, cleaner code

Questions

- GitHub Repository
  - `box_config` source code
  - H.A.C.K.S. proofs of concept
  - Reworked UVM 1.2 UBus example
  - Reusable `immutable_object` base class

https://github.com/williaml33moore/immutables

Thank you!