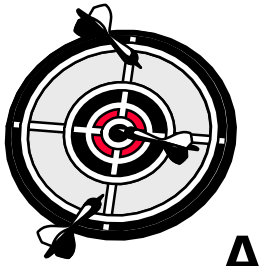
Agenda: Day 1

DAY 1

1. OOP Inheritance Review
2. UVM Structural Overview 
3. UVM Transaction
4. UVM Sequence 

Unit Objectives



After completing this unit, you should be able to:

- **Use OOP inheritance to create new OOP classes**
- **Use Inheritance to add new properties and functionalities**
- **Override methods in existing classes with inherited methods using virtual methods and polymorphism**

Object Oriented Programming (OOP): Class

- **Similar to a module, an OOP *class* encapsulates:**
  - Variables (properties) used to model a system
  - Subroutines (methods) to manipulate the data
  - Properties & methods are called members of class

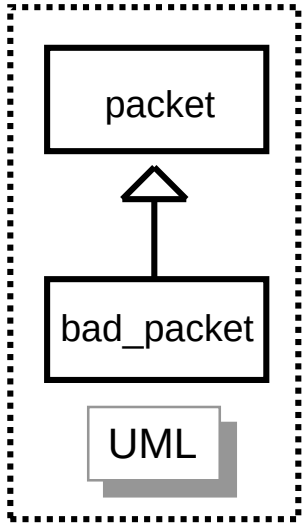
```
class packet;
  bit[3:0]  sa, da;            // packet class properties
  bit[7:0]  payload[$];        // packet class property
  int       crc;               // packet class property

  function int get_crc();     // packet class method
    ...
  endfunction

  function int compute_crc(); // packet class method
    ...
  endfunction
endclass: packet
```

Object Oriented Programming: Inheritance

- **OOP inheritance**
  - New classes extends from original (base) class
  - Inherits all contents of base class




```
class bad_packet extends packet;
```

packet

da
sa
payload
crc

```
function int get_crc();
  ...
endfunction
```

```
function int compute_crc();
  ...
endfunction
```

bad_packet

bad

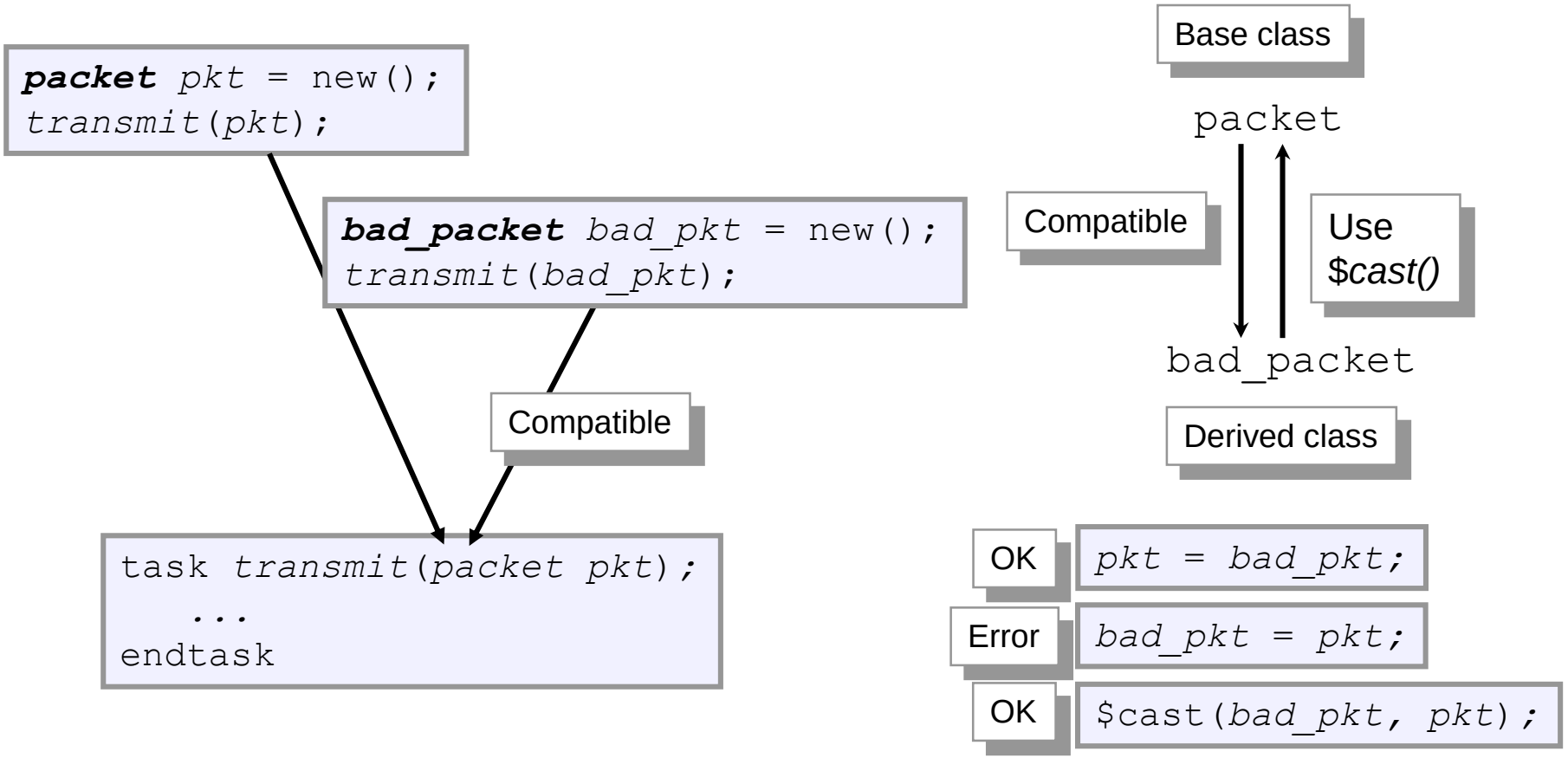
```
function int compute_crc();
  this.crc = super.compute_crc();
  if (bad) crc = ~crc;
endfunction
```

Can override methods in base class

Can call overridden method

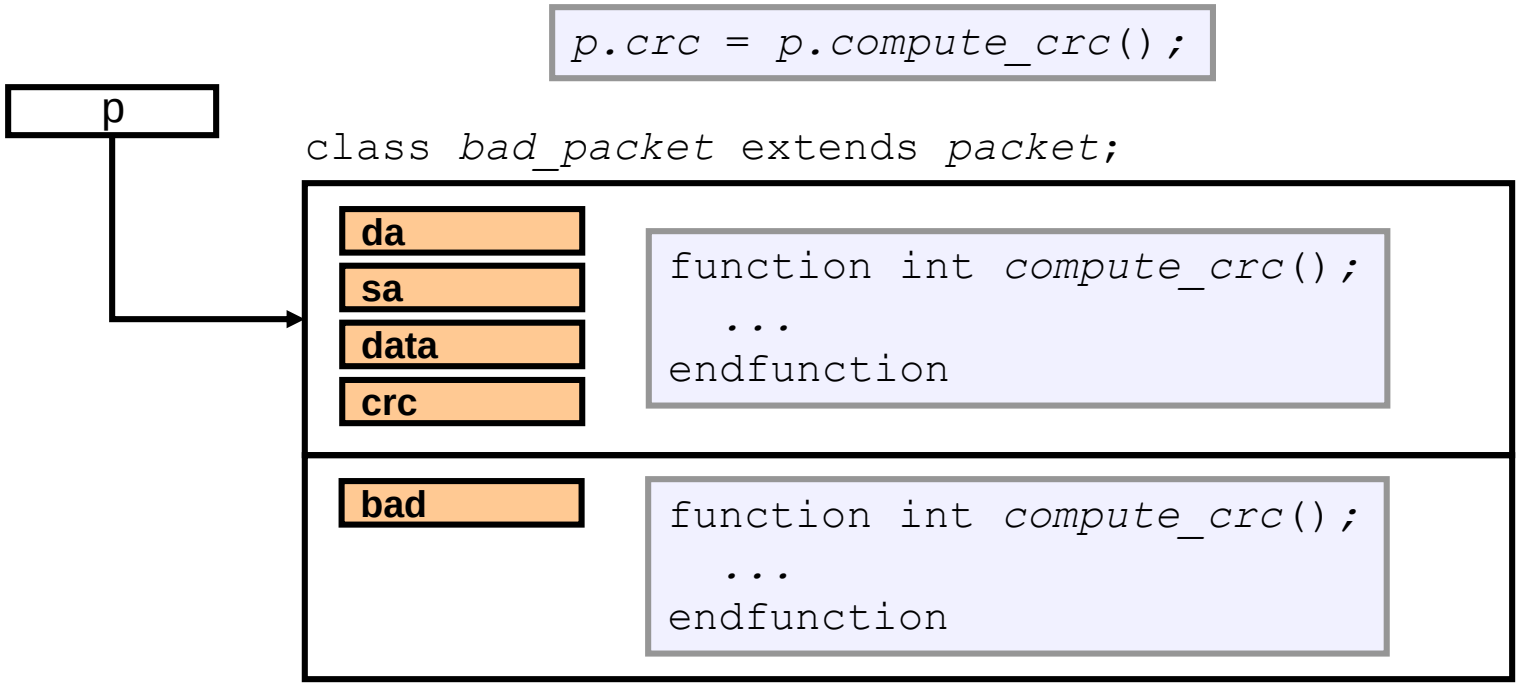
Object Oriented Programming: Inheritance

- **Derived classes compatible with base class**
  - Can reuse code



OOP: Polymorphism

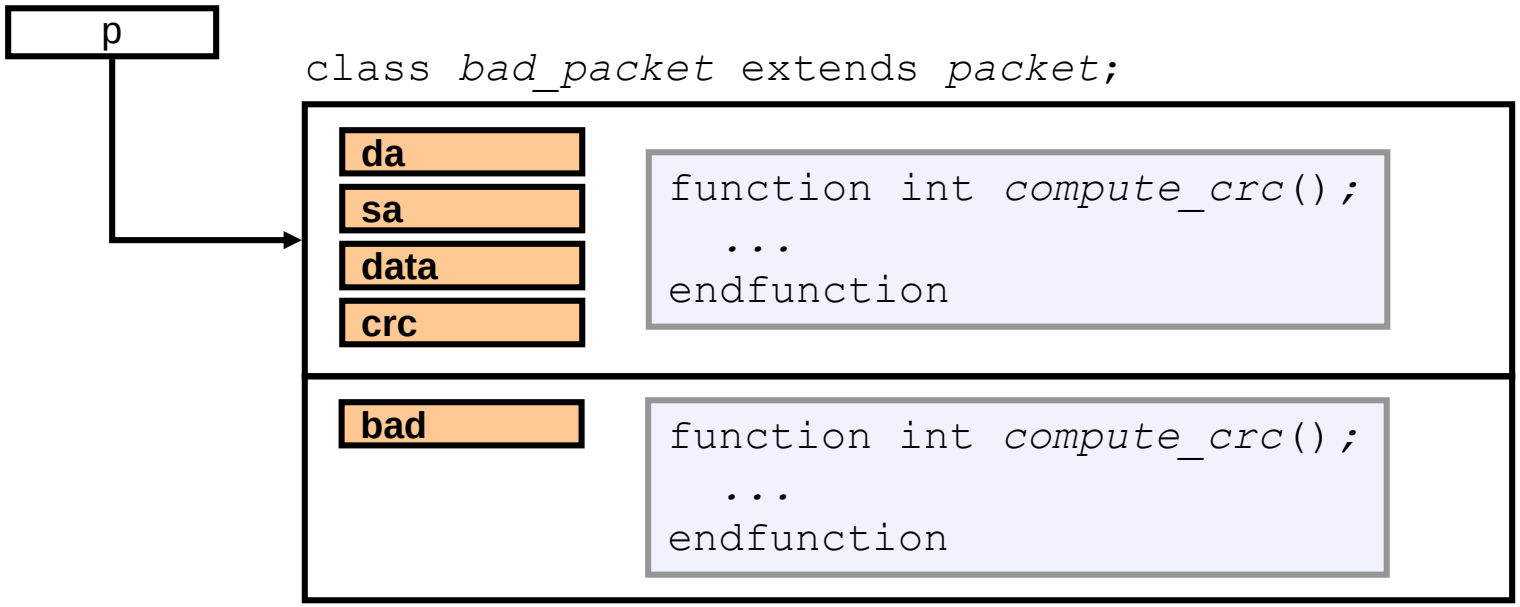
- **Which method gets called?**



- **Depends on**
  - Type of handle *p* (e.g. "***packet***" or "***bad_packet***" ?)
  - Whether ***compute_crc()*** is **virtual** or not

OOP: Polymorphism

- If *compute_crc()* is **not** **virtual**

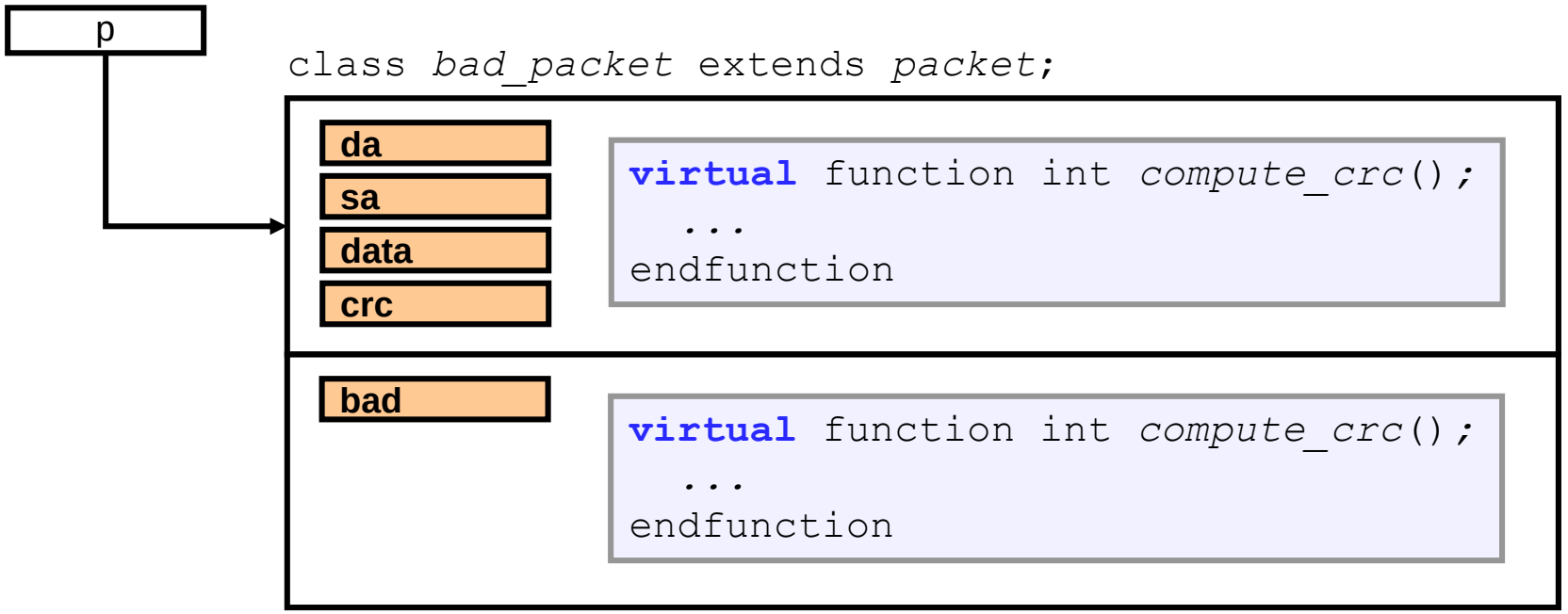



```
packet p = new();
bad_packet bp = new();
p.crc  = p.compute_crc();
bp.crc = bp.compute_crc();
transmit(p);
transmit(bp);
```

```
task transmit(packet pkt);
  pkt.crc = pkt.compute_crc();
  ...
endtask
```

OOP: Polymorphism

- If *compute_crc*() is virtual




```
packet p = new();
bad_packet bp = new();
p.crc  = p.compute_crc();
bp.crc = bp.compute_crc();
transmit(p);
transmit(bp);
```

```
task transmit(packet pkt);
  pkt.crc = pkt.compute_crc();
  ...
endtask
```

OOP: Polymorphism

- **Trying to inject errors**

```
protocol bfm = new(...);
repeat (100) begin
  bad_packet bp = new();
  bfm.transmit(bp);
end
```

```
class protocol;
  ...
  virtual task transmit(packet pkt);
    pkt.crc = pkt.compute_crc();
    ...
  endtask
endclass
```

pkt

class *bad_packet* extends *packet*;

compute_crc()

compute_crc()

If **virtual**

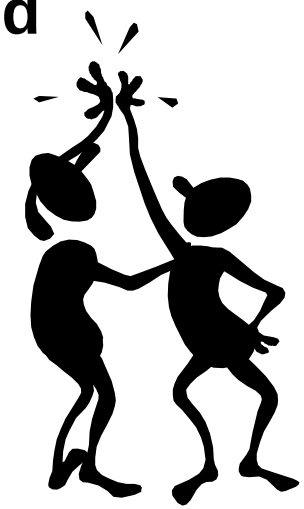
Guideline: methods should be `virtual`

Can inject CRC errors without modifying original code

Unit Objectives Review

You should now be able to:

- **Use OOP inheritance to create new OOP classes**
- **Use Inheritance to add new properties an functionalities**
- **Override methods in existing classes with inherited methods using virtual methods and polymorphism**



Appendix

Parameterized Class

Typedef Class

External Definition

Static Property and Method

Singleton Classes

Singleton Objects

Proxy Classes

Factory Class

Singleton Core Service Class

Parameterized Classes

- **Written for a generic type**
  - Type parameter passed at instantiation, just like parameterized modules
  - Allows reuse of common code

```
program automatic test;
  stack #(bit[31:0]) addr_stack;
  stack #(packet)    pkt_stack;
initial begin
  ...
  repeat(10) begin
    packet pkt = new();
    if(!pkt.randomize())
      $finish;
    pkt.addr = addr_stack.pop();
    pkt_stack.push(pkt);
  end
end
endprogram: test
```

```
class stack #(type T = int);
 local T items[$];
 function void push( T a );
 ...
 function T pop( );
 function int size(); ...
endclass
```

typedef

- **Can be used to make a forward declaration of a class**
  - e.g. two classes need handle to each other

```
typedef class S2;
class S1;
  S2 inside_s1;
  ...
endclass: S1
class S2;
  S1 i_am_inside;
  ...
endclass: S2
```

This is a compile error if typedef is missing

- **Or, to simplify parameterized class usage**

```
typedef stack #(bit[31:0]) stack32;
typedef stack #(packet)    stack_pkt;
program automatic test;
  stack32     addr_stack;
  stack_pkt   data_stack;
```

Methods Outside of the Class

- **The body of the class should fit on one “screen”**
  - Show the properties, and method headers
- **Method bodies can go later in the file**
  - Scope resolution operator **::** separates class and method name

```
class packet;
  bit[3:0] da, sa; bit[7:0] payload[$]; int crc;
  extern virtual function int get_crc();
  extern virtual function int compute_crc();
endclass
function int packet::get_crc();
  ...
endfunction
function int packet::compute_crc();
   ...
endfunction
```

Static Property

- **How do I create a variable shared by all objects of a class, but not make a global?**
- **A static property is associated with the class, not the object**
  - Can store meta-data, such as number of instances constructed
  - Single memory allocated all objects of that class
    - All objects share access to same memory

```
class packet;
  static int count = 0;
  int id;
  function new();
    id = count++;
  endfunction
endclass
```

Using a `id` field can help keep track of transactions as they flow through test

Static Method

- **A static method allows access via class name**
  - Can only access static properties in the class
  - Method memory is allocated per call
    - Each object static method call creates its own memory
- **Cannot be declared as `virtual`**

```
class packet;
  static int count = 0;
  int id;
  static function void print_count();
    $display("Created %0d packets", count);
  endfunction
  …
endclass
```

```
function void test::end_of_test();
  packet::print_count();
  ...
endfunction
```

Singleton Classes

- **Used to define a "global" service activity such as printing**
- **No object of the singleton class exists/allowed**
- **Contains only static members**

```
class print;
  static int err_count = 0, max_errors = 10;
  static function void error (string msg);
    $display("@%t: ERROR %s", $realtime, msg);
    if (err_count++ > max_errors)
      $finish;
  endfunction
  protected function new();
  endfunction
endclass
```

All members are static

Static methods cannot be virtual

No object allowed to exist

```
if (expect != actual)
  print::error("Actual did not match expected");
```

Singleton Objects

- **A singleton object is a globally accessible static object providing customizable service methods**
  - One and only one object in existence
    - Created at compile-time
  - Globally accessible at run-time
  - Can have static and non-static members
  - See slides 25 & 26 for customization

```
class service_class;
  protected static service_class me = get();
  static function service_class get();
    if (me == null) me = new(); return me;
  endfunction
  extern virtual function void error (string msg);
endclass
```

Object created at compile-time

Globally accessible at run-time

Non-static & virtual

```
service_class service_object = service_class::get();
service_object.error("A different error");
```