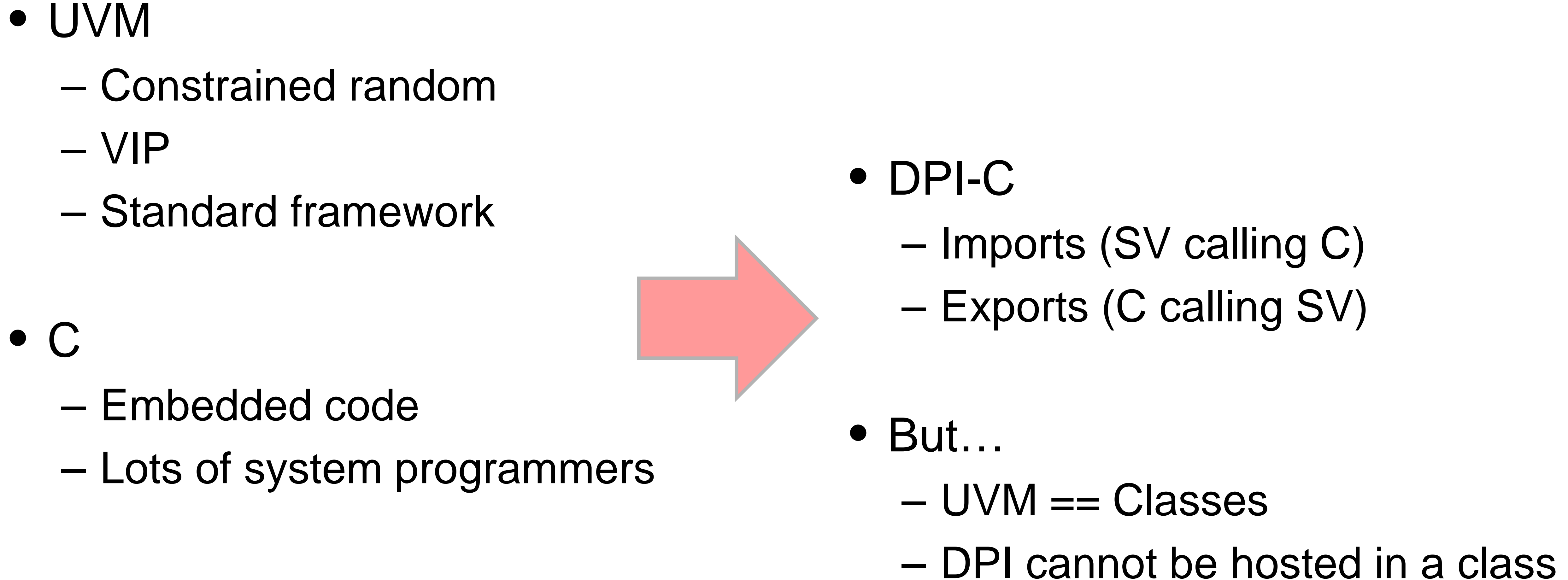
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

UVM and C – Perfect Together

Rich Edelman
Mentor, A Siemens Company
Fremont, CA

Mentor® A Siemens Business

UVM and C – Perfect Together, But…



Scope → Rules we don't like

- DPI requires a DPI import or DPI export statement.
- They cannot be in a class → There's no CLASS scope.

- They MUST be in module instance scope / package scope or file scope..

- But….We're doing everything in Class based code

SV DPI-C – A Quick Review

- Prototype statements
- Imports
- Exports
- Scope
- Data types

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2018

DPI Scope – test.sv

```
package c_pkg;
  import "DPI-C" function void c_package_scope();
endpackage

interface my_interface;
  import "DPI-C" function void c_interface_scope();
endinterface

import "DPI-C" function void c_file_scope();
import c_pkg::*;

class C;
  virtual my_interface vif;
  // ** Error: (vlog-13069) tb.sv(16): …
  // import "DPI-C" function void c_class_scope();
  function void print();
    c_package_scope();
    c_file_scope();
    vif.c_interface_scope();
    top.c_module_scope();
  endfunction
endclass
```

```
module top();
  C c;
  my_interface my_interface_instance();
  import "DPI-C" function
     void c_module_scope();

  initial begin
    c_package_scope();
    c_file_scope();
    c_module_scope();
    c = new();
    c.vif = my_interface_instance;
    c.print();
  end
endmodule
```

```
#  Hello! from c_pkg.
#  Hello! from tb_sv_unit.
#  Hello! from top.
#  Hello! from c_pkg.
#  Hello! from tb_sv_unit.
#  Hello! from top.my_interface_instance.
#  Hello! from top.
```

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

DPI Scope – ccode.c

- Just print the SCOPE

```
void c_package_scope() {
  const char *scopeName = svGetNameFromScope(svGetScope());
  printf(" Hello! from %s.\n", scopeName);
}
void c_file_scope() {
  const char *scopeName = svGetNameFromScope(svGetScope());
  printf(" Hello! from %s.\n", scopeName);
}
void c_class_scope() {
  const char *scopeName = svGetNameFromScope(svGetScope());
  printf(" Hello! from %s.\n", scopeName);
}
void c_interface_scope() {
  const char *scopeName = svGetNameFromScope(svGetScope());
  printf(" Hello! from %s.\n", scopeName);
}
void c_module_scope() {
  const char *scopeName = svGetNameFromScope(svGetScope());
  printf(" Hello! from %s.\n", scopeName);
}
```

SV / C Interface – Prototype Statements

```
module top();

  import "DPI-C" context task c_hello(int inst_id, int i, int j, output int k);
  export "DPI-C"         task sv_hello;


  task automatic sv_hello(int inst_id);
    #5;
  endtask

  initial begin
    int k;
    c_hello(1, 2, 3, k);
  end
```

SV

```
#include "svdpi.h"

int
c_hello(
    int inst_id,
    int i,
    int j,
    int* k);

int
sv_hello(
    int inst_id);
```

C

dpiheader.h

SV → C – IMPORT +threads+scope

SV

```
initial begin
  int k;
  c_hello(1, 2, 3, k); //  6
  ...
end
```

&k
3
2
1

```
initial begin
  int k;
  c_hello(2, 5, 6, k); // 30
  ...
end
```

&k
6
5
2

C

```
#include <stdio.h>
#include "dpiheader.h"

int global_id;

int
c_hello(int inst_id, int i, int j, int *k)
{
  const char *scopeName =
    svGetNameFromScope(svGetScope());

  global_id = inst_id;

  *k = i + j;

  if (global_id != inst_id)
    printf(" c...");
  *k = i * j;
  return 0;
}
```

SV → C → SV – IMPORT & EXPORT

SV

```
initial begin
  int k;
  c_hello(1, 2, 3, k); //  6
  ...
end
```

&k
3
2
1

```
initial begin
  int k;
  c_hello(2, 5, 6, k); // 30
  ...
end
```

&k
6
5
2

C

```
#include <stdio.h>
#include "dpiheader.h"

int global_id;

int
c_hello(int inst_id, int i, int j, int *k)
{
  const char *scopeName =
    svGetNameFromScope(svGetScope());

  global_id = inst_id;

  *k = i + j;
  sv_hello(inst_id);
  if (global_id != inst_id)
    printf(" c...");
  *k = i * j;
  return 0;
}
```

```
task sv_hello(int inst_id);
  #5;
endtask
```

Scope → What's My Scope?

- Module instance scope

```
int
c_hello(int inst_id, int i, int j, int *k)
{
    const char *scopeName = svGetNameFromScope(svGetScope());
    printf(" c:       %d Hello! from %s.\n", inst_id, scopeName);
```

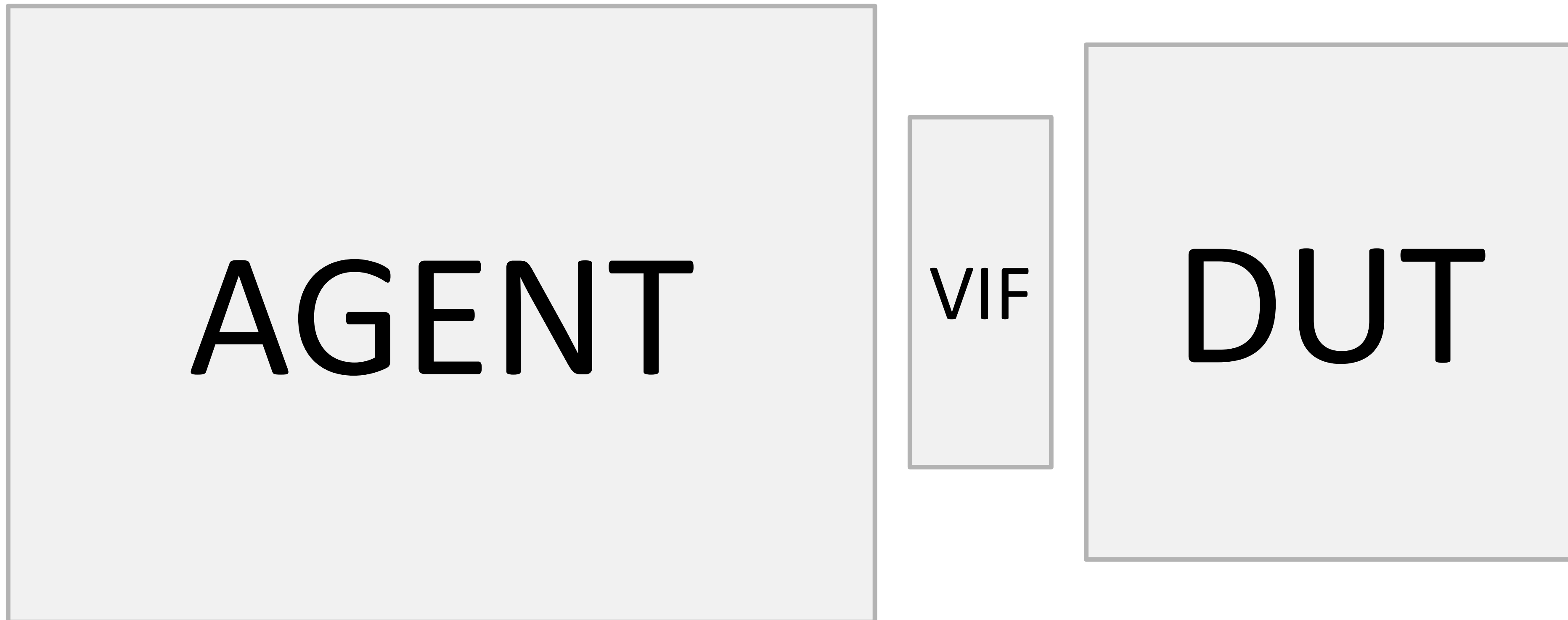
```
module top();
  task automatic sv_hello(int inst_id);
    $display("sv: @%0t: %0d Hello! from %m.", $time, inst_id);
    #5;
  endtask
```

Data types → Rules we made for ourselves

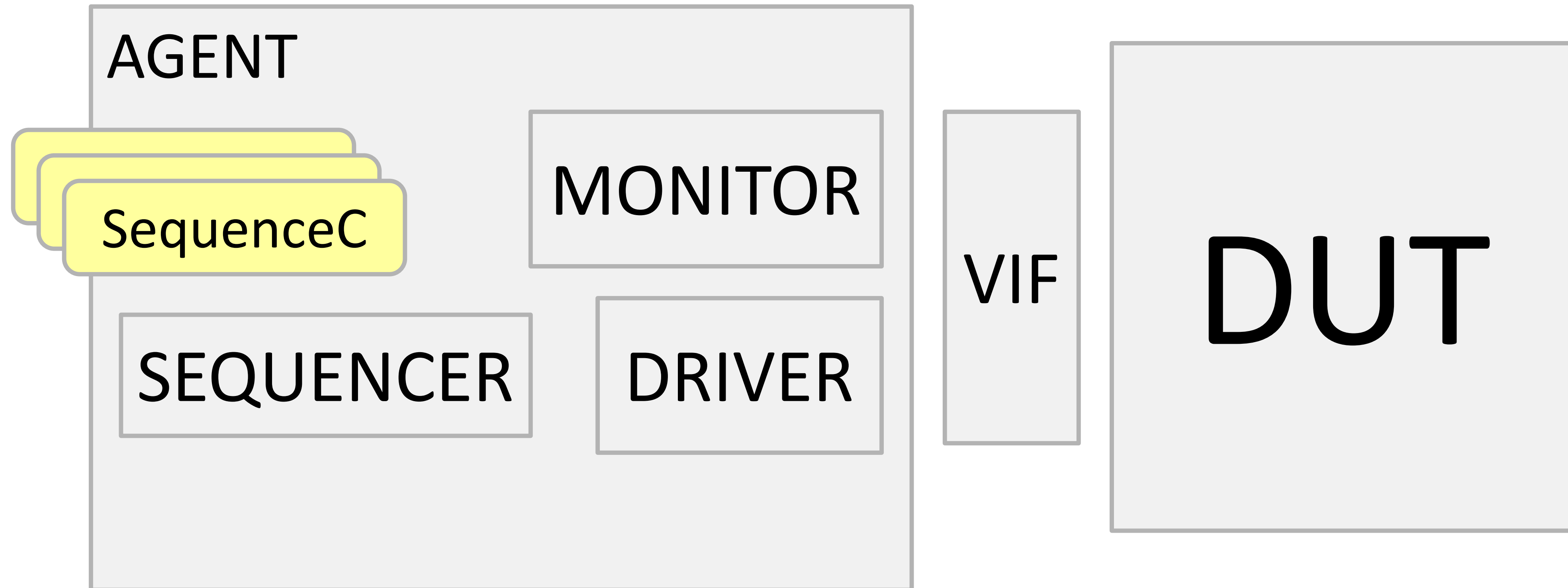
- Let C do things it is good at
  - integers, bits, simple arrays
- Let SV do things it is good at
  - 4 state logic, long bit vectors, associative arrays, dynamic arrays

The UVM Agent – Quick Review

- For every agent, one interface

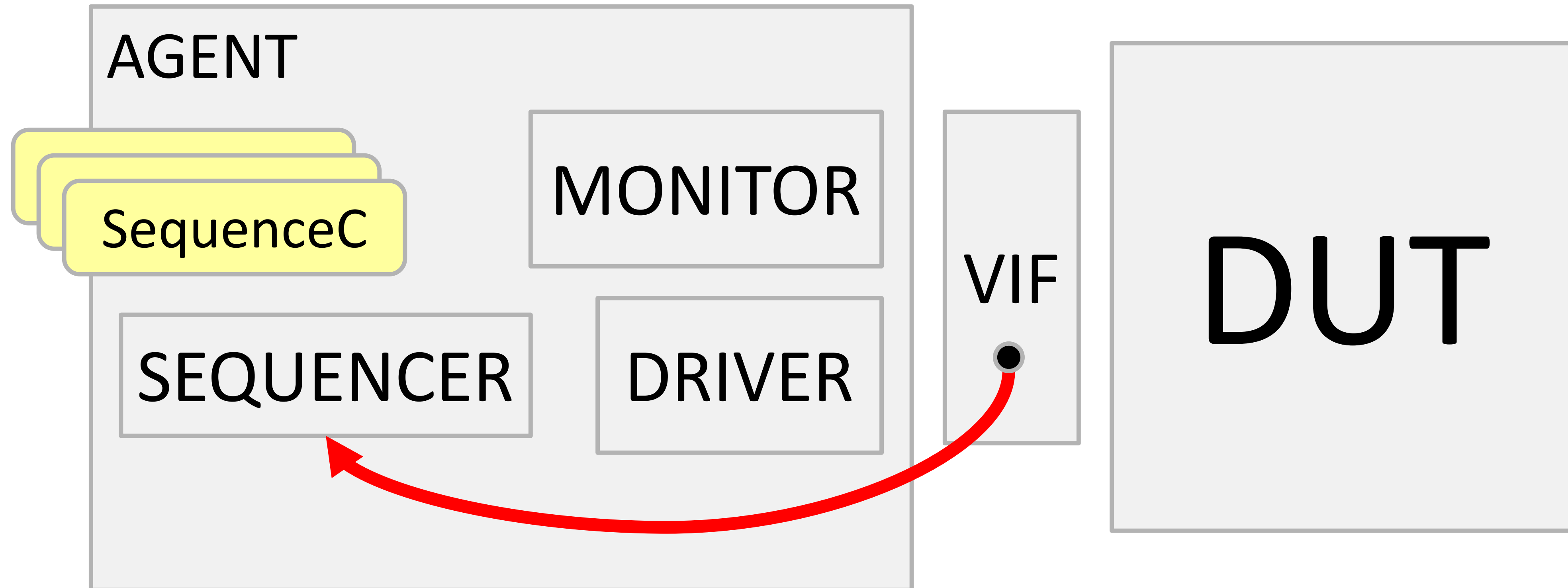


The UVM Agent – Details



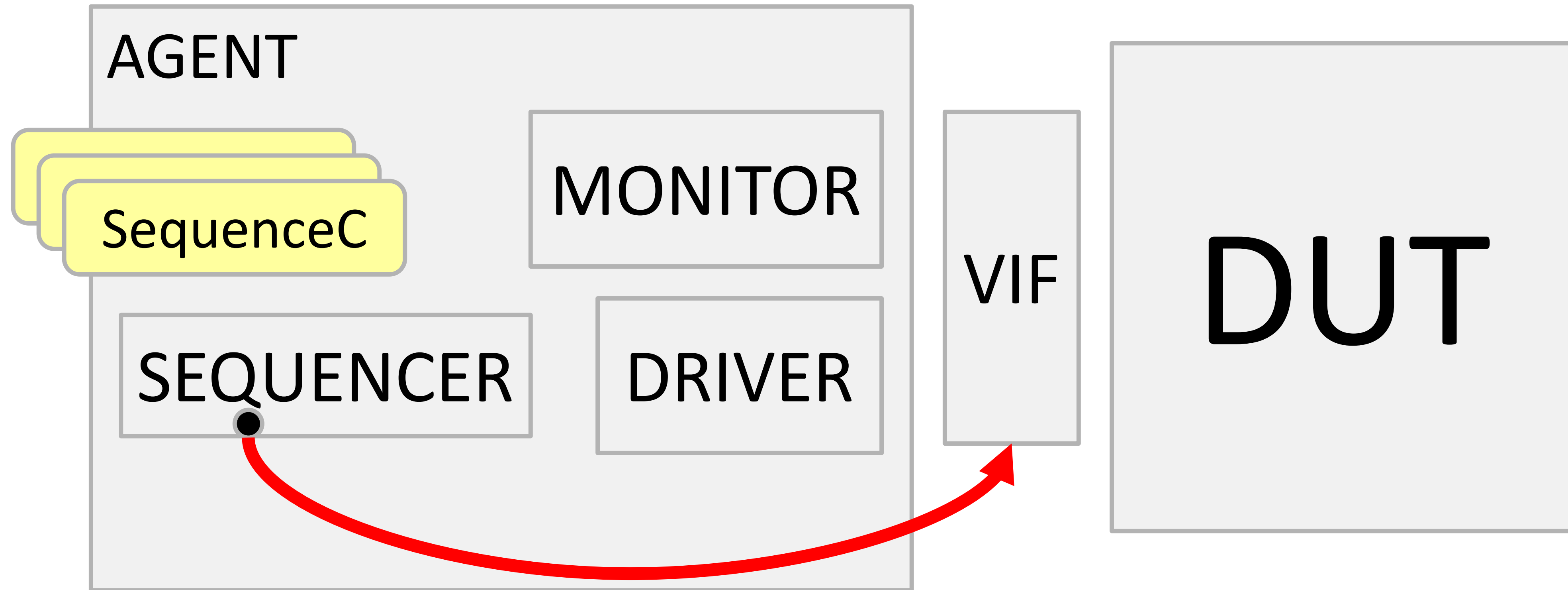
An Interface that is "Sequencer-aware"

- Call sequencer code through the interface



A Sequencer that is “Interface-aware”

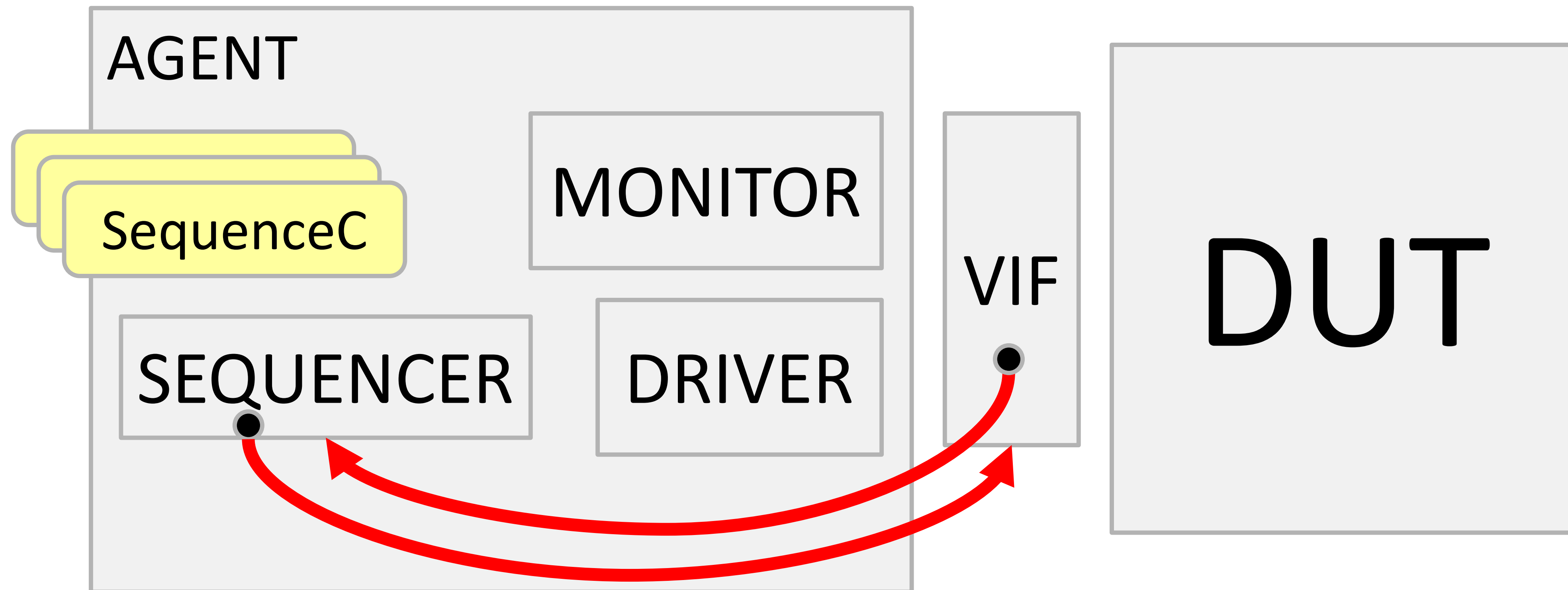
- Call interface code through the sequencer



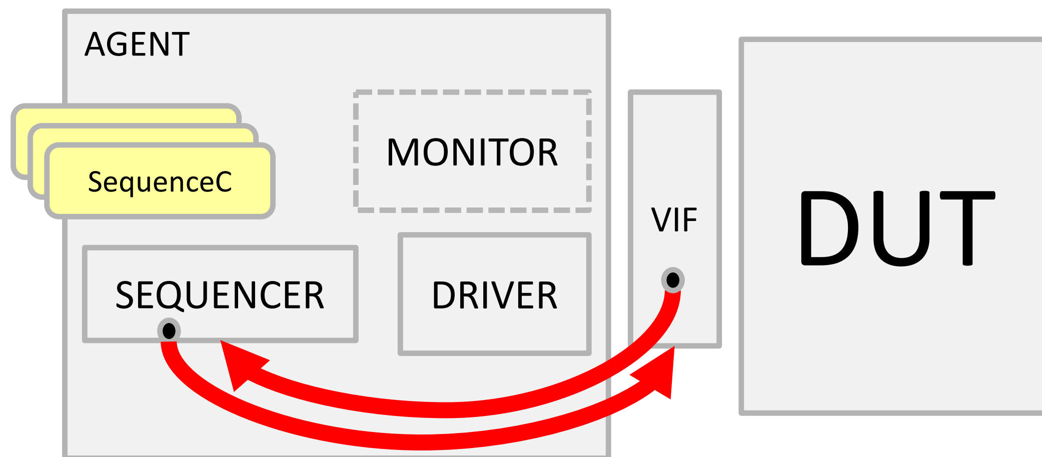
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Sequencer ←→ Interface

- Lots of sequences - All work is done with sequences



The Agent




```
class agent extends uvm_agent;
  `uvm_component_utils(agent)

  driver      d;
  sequencer sqr;

  virtual my_interface vif;

  function void build_phase(uvm_phase phase);
    d = driver::type_id::create("d", this);
    sqr = sequencer::type_id::create("sqr", this);
    vif.sqr = sqr;
    sqr.vif = vif;
  endfunction

  function void connect_phase(uvm_phase phase);
    d.seq_item_port.connect(sqr.seq_item_export);
  endfunction
endclass
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2018

The SystemVerilog Virtual Interface

```
interface my_ahb_bus(wire clk, …);
  <wires>









endinterface
```

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

The SystemVerilog Virtual Interface

```
interface my_ahb_bus(wire clk, …);
  <wires>





  uvm_sequencer_base sqr;









endinterface
```

The SystemVerilog Virtual Interface

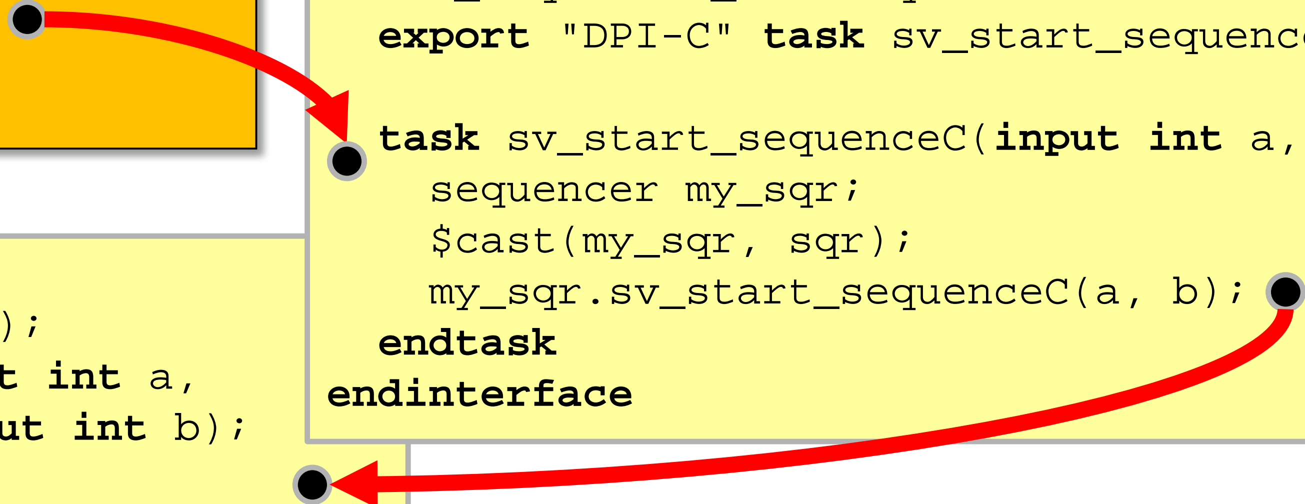
```
interface my_ahb_bus(wire clk, …);
  <wires>
  import "DPI-C" context function void c_hello (input int id);
  import "DPI-C" context task          c_thread(input int id);
  export "DPI-C"         function      sv_hello;
  export "DPI-C"         task          sv_start_sequenceC;

  uvm_sequencer_base sqr;

task sv_start_sequenceC(input int a, output int b);
   sequencer my_sqr;
   $cast(my_sqr, sqr);
   my_sqr.sv_start_sequenceC(a, b);
endtask

function void sv_hello(input int id);
   sequencer my_sqr;
   $cast(my_sqr, sqr);
   my_sqr.sv_hello(id);
endfunction

endinterface
```

A Sequencer that is "DPI-aware"

```
class sequencer extends uvm_sequencer#(transaction);
  virtual my_interface vif;

  function void sv_hello(int inst_id);
    $display("sv: Hello! from %s. inst_id=%0d", get_full_name(), inst_id);
  endfunction

  task c_start_threads();
    fork
      for (int i=0; i<4; i++) vif.c_thread(...);
    join
  endtask

  task sv_start_sequenceC(input int a, output int b);
    sequenceC seq;
    seq = sequenceC::type_id::create("seqC");
    seq.a = a;
    seq.start(this);
    b = seq.b;
  endtask
endclass
```

C code calling SV

- C code should cause a sequence to execute on the sequencer



```c
int c_thread(int id)
{
  int b;
  sv_start_sequenceC(id, &b);
  return 0;
}
```

```systemverilog
typedef class sequencer;
interface my_interface(input clk);
  uvm_sequencer_base sqr;
  export "DPI-C" task sv_start_sequenceC;

  task sv_start_sequenceC(input int a, output int b);
    sequencer my_sqr;
    $cast(my_sqr, sqr);
    my_sqr.sv_start_sequenceC(a, b);
  endtask
endinterface
```

```systemverilog
class sequencer extends
    uvm_sequencer#(transaction);
  task sv_start_sequenceC(input int a,
                          output int b);
    sequenceC seq;
    seq = sequenceC::type_id::create("seqC");
    seq.a = a;
    seq.start(this);
    b = seq.b;
  endtask
endclass
```

SV code calling C

- SV code should call C code through a virtual interface

```
class sequenceA extends uvm_sequence#(transaction);
  int a, b;
  virtual my_interface vif;

  task body();
    sequencer sqr;
    int i[], o[], io[];
    $cast(sqr, m_sequencer);
    vif = sqr.vif;
    vif.c_hello(get_inst_id());

    vif.c_datatype_openarray_of_int(i, o, io);
```

```
interface my_interface(input clk);
  import "DPI-C" context function void c_datatype_openarray_of_int(
    input int i[],
    output int o[],
    inout int io[]);
endinterface
```

```
void
c_datatype_openarray_of_int(
    const svOpenArrayHandle i, o, io){
  int *from_i, *to_o;

  from_i  = svGetArrayPtr(i);
  to_o    = svGetArrayPtr(o);

  for (int j=0; j<svSize(i,1); j++) {
    *to_o++  = *from_i++;
  }
}
```

Fine. What do I have to change to get hooked up?

- Sequencer gets a VIF handle
- VIF gets a sequencer handle
- VIF defines imports and exports
- VIF contains helper functions (if needed)
- Sequencer contains helper functions (if needed)

Sequencer gets a VIF handle

```
class sequencer extends uvm_sequencer#(transaction);
  virtual my_interface vif;

  task c_start_threads();
    fork
      for (int i=0; i<4; i++) vif.c_thread(...);
    join
  endtask
endclass
```

VIF gets a sequencer handle

```
interface my_ahb_bus(wire clk, …);
  <wires>

  uvm_sequencer_base sqr;

endinterface
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2018 UNITED STATES

VIF defines imports

```
interface my_ahb_bus(wire clk, …);
  <wires>
  import "DPI-C" context function void c_hello (input int id);
  import "DPI-C" context task          c_thread(input int id);


  uvm_sequencer_base sqr;










endinterface
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2018 UNITED STATES

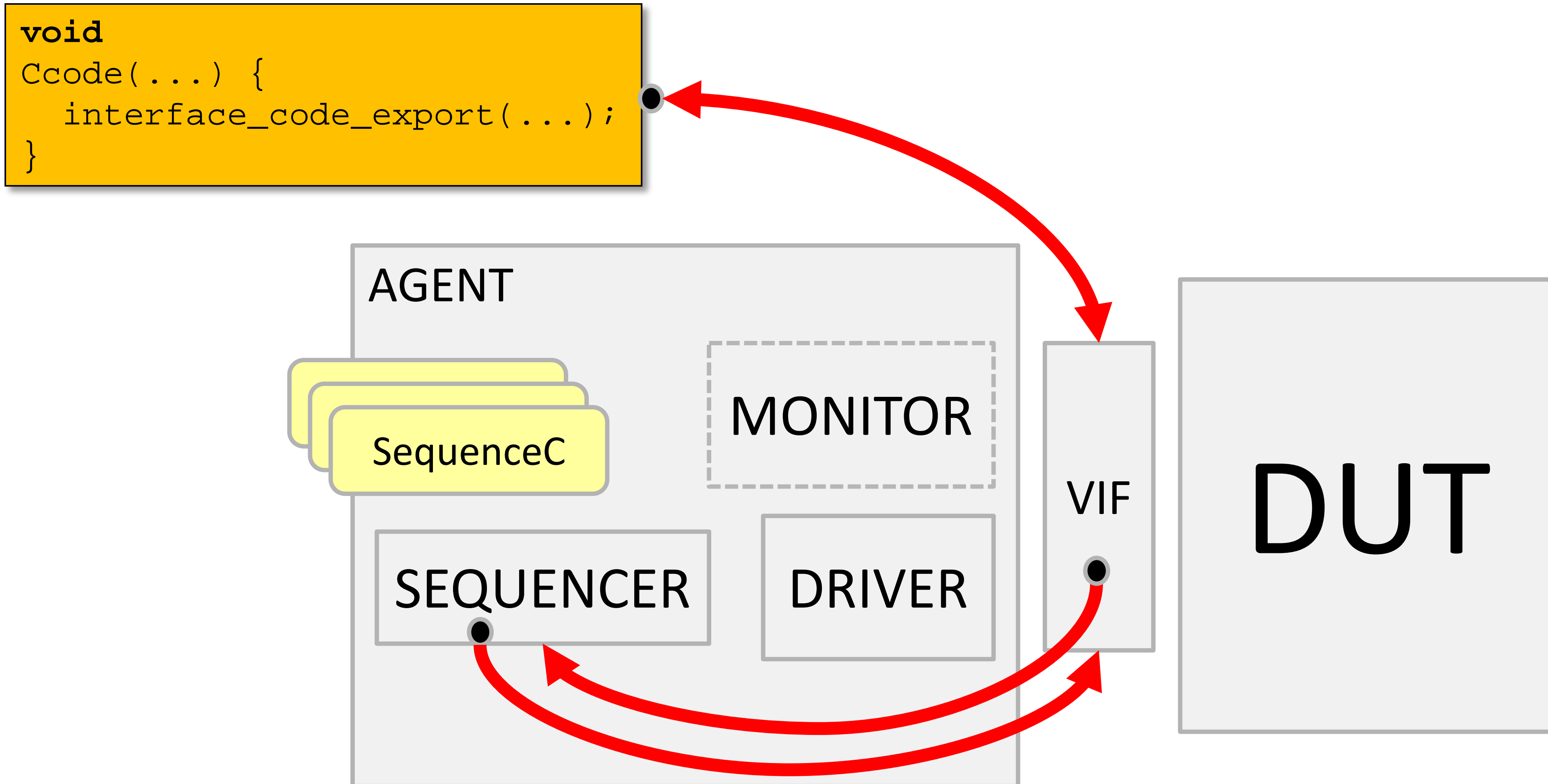
VIF contains export helper functions

```
interface my_ahb_bus(wire clk, …);
  <wires>


  export "DPI-C"           function       sv_hello;
  export "DPI-C"           task           sv_start_sequenceC;

  uvm_sequencer_base sqr;

task sv_start_sequenceC(input int a, output int b);
   sequencer my_sqr;
   $cast(my_sqr, sqr);
   my_sqr.sv_start_sequenceC(a, b);
endtask

function void sv_hello(input int id);
   sequencer my_sqr;
   $cast(my_sqr, sqr);
   my_sqr.sv_hello(id);
endfunction

endinterface
```

Sequencer contains helper functions

```
class sequencer extends uvm_sequencer#(transaction);
  virtual my_interface vif;

  function void sv_hello(int inst_id);
    $display("sv: Hello! from %s. inst_id=%0d", get_full_name(), inst_id);
  endfunction

  task c_start_threads();
    fork
      for (int i=0; i<4; i++) vif.c_thread(...);
    join
  endtask

  task sv_start_sequenceC(input int a, output int b);
    sequenceC seq;
    seq = sequenceC::type_id::create("seqC");
    seq.a = a;
    seq.start(this);
    b = seq.b;
  endtask
endclass
```

C ← → Agent



Summary

- Using Sequencer and Interface allows a standard way to
  - Call C from SV
  - Call SV from C
- C calls become sequences and sequence items
  - Integrating completely with other sequence based traffic on the driver
- DPI provides an easy to use, high performance interface to UVM
  - And an easy way for SV and C to interoperate

rich_edelman@mentor.com for source code or questions