University of Zurich[UZH]

Functional-style programming with Fortran 2003

CP2K User Meeting 2017 — Developers Day

Tiziano Müller

3. July 2017

Dept. of Chemistry, UZH

Interface Problem

```
SUBROUTINE func(..)
   ! setup

   ! do ...
   !    operation
   ! end do

   ! cleanup
END SUBROUTINE func
```

Examples:

- Make function to be integrated interchangeable
- Multiple functions to be applied “for each” element
- Outer-function is in principal independant of inner-function parameters

The example

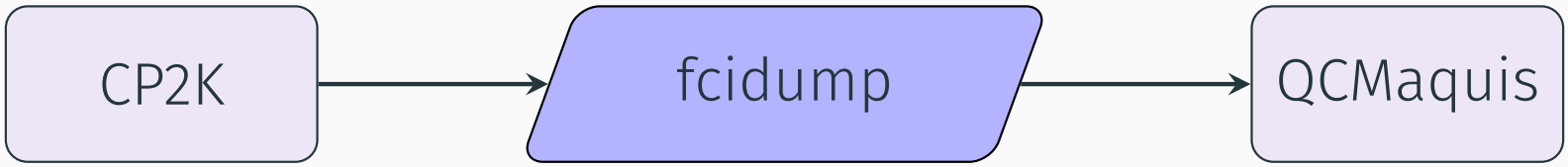


Figure 1: Interfacing CP2K and QCMaquis via file

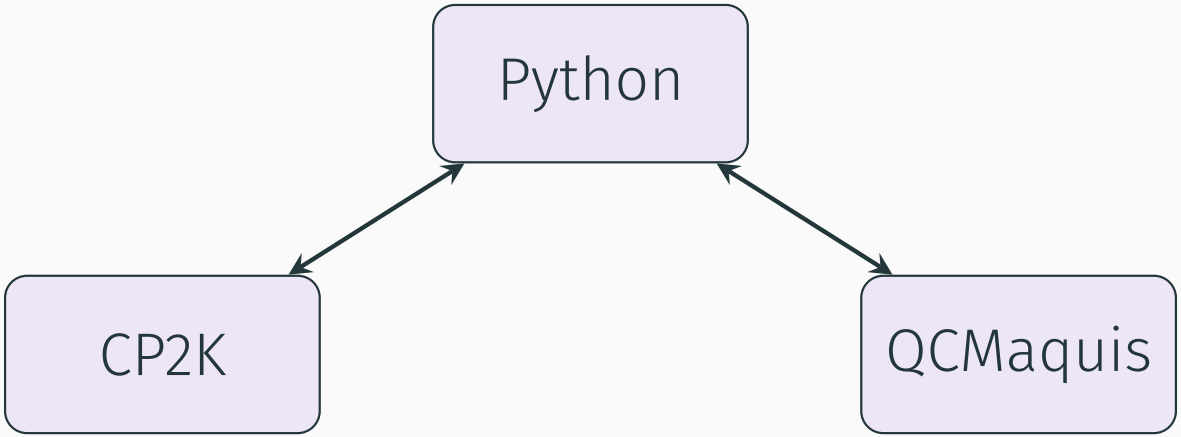
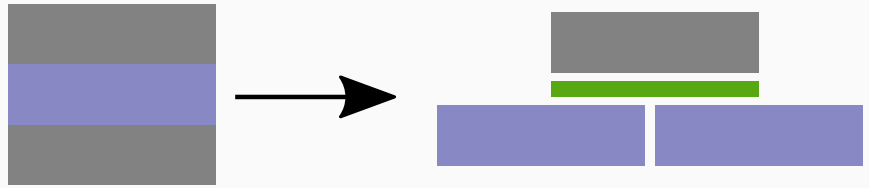


Figure 2: Interfacing CP2K and QCMaquis via Python

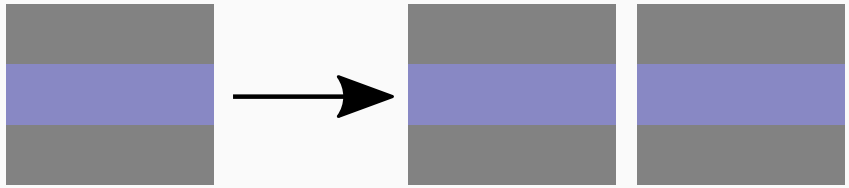
The example (code)

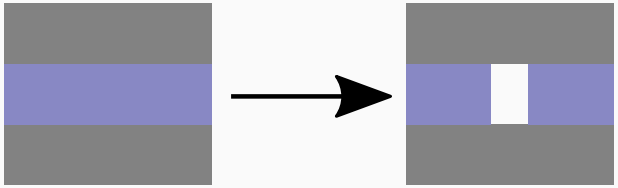
```
SUBROUTINE fcidump(active_space_env, as_input)
   ! [...]
   CALL mp_sum(nindex, eri%mp_group)
   CALL mp_sum(colind(1:nindex), eri%mp_group)
   CALL mp_sum(erival(1:nindex), eri%mp_group)
   CALL mp_sync(eri%mp_group)
   IF (iw > 0) THEN
      DO i34l = 1, nindex
         i34 = colind(i34l)
         erint = erival(i34l)
         CALL get_index_pair(i34, norb, i3, i4)
         WRITE (iw,"(ES23.16,4I4)") erint,i1,i2,i3,i4
      END DO
   END IF
   ! [...]
END SUBROUTINE fcidump
```

Legacy solutions



- Split function
- Intermediate storage structures
- Code replication
- Preprocessor-based includes
- Case-Switch





Procedure pointers

```
MODULE my_mod
  INTERFACE
    SUBROUTINE my_proc()
       ! [...]
    END SUBROUTINE my_proc
  END INTERFACE
CONTAINS
  SUBROUTINE my_proc1()
    ! [...]
  END SUBROUTINE my_proc1
  SUBROUTINE my_proc2()
    ! [...]
  END SUBROUTINE my_proc2

  SUBROUTINE call_proc(func_ptr)
    PROCEDURE (my_proc), POINTER
     ↪  :: func_ptr
    CALL func_ptr()
  END SUBROUTINE call_proc
END MODULE my_mod
```

```
PROGRAM my_test
   USE my_mod, ONLY:
    ↪  my_proc, my_proc1,
    ↪  my_proc2, call_proc

   PROCEDURE (my_proc),
    ↪  POINTER :: func_ptr
    ↪  => null()
   ! [...]

   func_ptr => my_proc1
   CALL call_proc(func_ptr)

   func_ptr => my_proc2
   CALL call_proc(func_ptr)
END PROGRAM my_test
```

Procedure pointers

```
MODULE my_mod
  INTERFACE
    SUBROUTINE my_proc()
      ! [...]
    END SUBROUTINE my_proc
  END INTERFACE
CONTAINS
  SUBROUTINE my
    ! [...]
  END SUBROUTIN
  SUBROUTINE my_proc2()
    ! [...]
  END SUBROUTINE my_proc2

  SUBROUTINE call_proc(func_ptr)
    PROCEDURE (my_proc), POINTER
     ↪  :: func_ptr
    CALL func_ptr()
  END SUBROUTINE call_proc
END MODULE my_mod
```

→What about function-specific data?

```
PROGRAM my_test
   USE my_mod, ONLY:
    ↪  my_proc, my_proc1,
    ↪  my_proc2, call_proc

                    _proc),
                    :: func_ptr
                    ()
   ! [...]

   func_ptr => my_proc1
   CALL call_proc(func_ptr)

   func_ptr => my_proc2
   CALL call_proc(func_ptr)
END PROGRAM my_test
```

Object-oriented Fortran

```
MODULE my_mod
  TYPE :: my_type
    INTEGER :: val

    CONTAINS
      PROCEDURE, PASS(self) :: func
  ↪   => my_type_func
  END TYPE my_type
CONTAINS
  SUBROUTINE my_type_func(self, val)
    CLASS(my_type), INTENT(inout) ::
      ↪   self
    INTEGER, INTENT(IN)           ::
      ↪   val

    self%val = val
  END SUBROUTINE my_type_func
END MODULE my_mod
```

```
PROGRAM my_test
   USE my_mod, ONLY:
     ↪   my_type

   type(my_type) :: var

   CALL var%func(10)
END PROGRAM my_test
```

Derived Types

+ Type-bound procedures

Interfaces and Polymorphism

```
MODULE my_mod
  TYPE, ABSTRACT :: abstract_type
  CONTAINS
    PROCEDURE(abstract_type_func), DEFERRED :: func
  END TYPE abstract_type

  ABSTRACT INTERFACE
    SUBROUTINE abstract_type_func(self, idx)
      IMPORT                        :: abstract_type
      CLASS(abstract_type), INTENT(inout) :: self
      INTEGER, INTENT(in)                 :: idx
    END SUBROUTINE abstract_type_func
  END INTERFACE

  TYPE, EXTENDS(abstract_type) :: type_impl1
    INTEGER :: idx
    CONTAINS
      PROCEDURE :: func => type_impl1_func
  END TYPE
CONTAINS
  SUBROUTINE type_impl1_func(self, idx)
    CLASS(type_impl1), INTENT(inout) :: self
    INTEGER, INTENT(IN)              :: idx

    self%idx = idx
  END SUBROUTINE type_impl1_func
END MODULE
```

```
PROGRAM my_test
  USE my_mod, ONLY:
    ↪   abstract_type,
    ↪   type_impl1

  type(type_impl1) :: var
  CALL do_smth(var)

CONTAINS
  SUBROUTINE do_smth(var)
    CLASS(abstract_type) ::
      ↪   var
    CALL var%func(10)
  END SUBROUTINE do_smth
END PROGRAM my_test
```

Abstract Types

+ Abstract Interfaces
+ Inheritance

The example revisited (code)

```
SUBROUTINE fcidump(active_space_env, as_input)
   ! [...]
   CALL mp_sum(nindex, eri%mp_group)
   CALL mp_sum(colind(1:nindex), eri%mp_group)
   CALL mp_sum(erival(1:nindex), eri%mp_group)
   CALL mp_sync(eri%mp_group)
   IF (iw > 0) THEN
      DO i34l = 1, nindex
         i34 = colind(i34l)
         erint = erival(i34l)
         CALL get_index_pair(i34, norb, i3, i4)
         WRITE (iw,"(ES23.16,4I4)") erint,i1,i2,i3,i4
      END DO
   END IF
   ! [...]
END SUBROUTINE fcidump
```

The example revisited (code)

```
SUBROUTINE fcidump(active_space_env, as_input)
   ! [...]
   CALL mp_sum(nindex, eri%mp_group)
   CALL mp_sum(colind(1:nindex), eri%mp_group)
   CALL mp_sum(erival(1:nindex), eri%mp_group)
   CALL mp_sync(eri%mp_group)
   IF (iw > 0) THEN
      DO i34l = 1, nindex
         i34 = colind(i34l)
         erint = erival(i34l)
         CALL get_index_pair(i34, norb, i3, i4)
         WRITE (iw,"(ES23.16,4I4)") erint,i1,i2,i3,i4
      END DO
   END IF
   ! [...]
END SUBROUTINE fcidump
```

The example revisited (code)

qs_active_space_types.F

```
TYPE eri_type
  ! [...]
  CONTAINS
    PROCEDURE :: eri_foreach => eri_type_eri_foreach
END TYPE eri_type

SUBROUTINE eri_type_eri_foreach(this, ispin, fobj)
   CLASS(eri_type)                  :: this
   CLASS(eri_type_eri_element_func) :: fobj
   ! [...]
END SUBROUTINE eri_type_eri_foreach

TYPE, ABSTRACT :: eri_type_eri_element_func
CONTAINS
  PROCEDURE(eri_type_eri_element_func_interface), DEFERRED :: func
END TYPE eri_type_eri_element_func

ABSTRACT INTERFACE
  LOGICAL FUNCTION eri_type_eri_element_func_interface(this, &
      i, j, k, l, val)
    IMPORT :: eri_type_eri_element_func, dp
    CLASS(eri_type_eri_element_func) :: this
    ! [...]
  END FUNCTION eri_type_eri_element_func_interface
END INTERFACE
```

The example revisited (code)

```
SUBROUTINE fcidump(active_space_env, as_input)
   ! [...]
   CALL mp_sum(nindex, eri%mp_group)
   CALL mp_sum(colind(1:nindex), eri%mp_group)
   CALL mp_sum(erival(1:nindex), eri%mp_group)
   CALL mp_sync(eri%mp_group)
   IF (iw > 0) THEN
      DO i34l = 1, nindex
         i34 = colind(i34l)
         erint = erival(i34l)
         CALL get_index_pair(i34, norb, i3, i4)
         WRITE (iw,"(ES23.16,4I4)") erint,i1,i2,i3,i4
      END DO
   END IF
   ! [...]
END SUBROUTINE fcidump
```

The example revisited (code)

qs_active_space_methods.F

SUI

```
TYPE, EXTENDS(eri_type_eri_element_func) :: eri_fcidump_print
  INTEGER :: unit_nr
  CONTAINS
    PROCEDURE :: func => eri_fcidump_print_func
END TYPE

LOGICAL FUNCTION eri_fcidump_print_func(this, i, j, k, l, val) RESULT(cont)
  CLASS(eri_fcidump_print), INTENT(inout) :: this
  INTEGER, INTENT(in)                     :: i, j, k, l
  REAL(KIND=dp), INTENT(in)               :: val

  ! write to the actual file only on the master
  IF (this%unit_nr > 0) THEN
     WRITE (this%unit_nr, "(ES23.16,4I4)") val, i, j, k, l
  END IF

  cont = .TRUE.
END FUNCTION eri_fcidump_print_func

! [...]
  IF (iw > 0) THEN
    CALL active_space_env%eri%eri_foreach(1, eri_fcidump_print(iw))
  END IF
! [...]
```

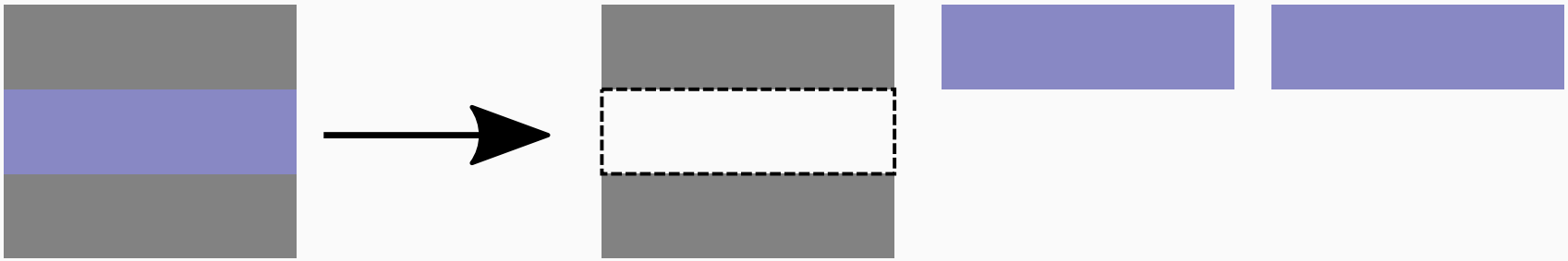
ENI

The example revisited (code)

start/libcp2k.F

```
TYPE, EXTENDS(eri_type_eri_element_func) :: eri2array
  INTEGER(C_INT), POINTER :: coords(:)
  REAL(C_DOUBLE), POINTER :: values(:)
  INTEGER                 :: idx = 1
  CONTAINS
    PROCEDURE :: func => eri2array_func
END TYPE

LOGICAL FUNCTION eri2array_func(this, i, j, k, l, val) RESULT(cont)
  CLASS(eri2array), INTENT(inout) :: this
  INTEGER, INTENT(in)             :: i, j, k, l
  REAL(KIND=dp), INTENT(in)       :: val

  this%coords(4*(this%idx-1) + 1) = i
  this%coords(4*(this%idx-1) + 2) = j
  this%coords(4*(this%idx-1) + 3) = k
  this%coords(4*(this%idx-1) + 4) = l
  this%values(this%idx)           = val

  this%idx = this%idx + 1

  cont = .TRUE.
END FUNCTION eri2array_func
```

The solution



→ Use OO Inheritance to vary behaviour

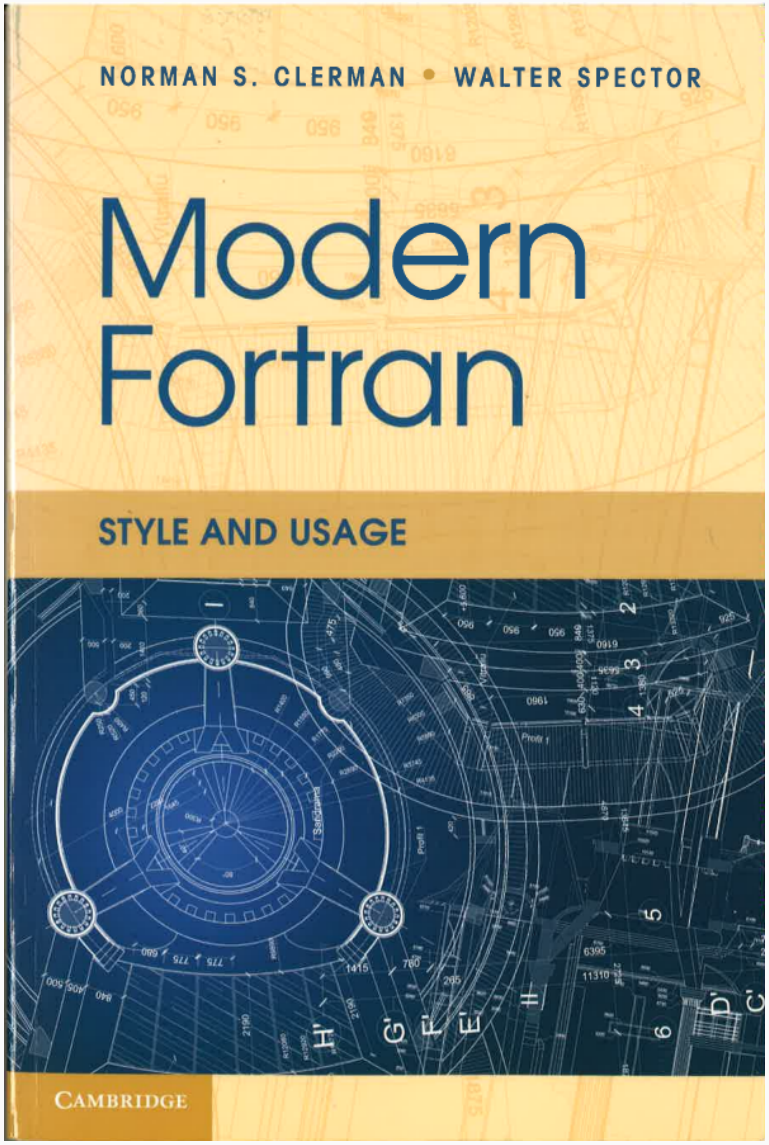
→ Use OO Derived Types to bind function parameters to procedures

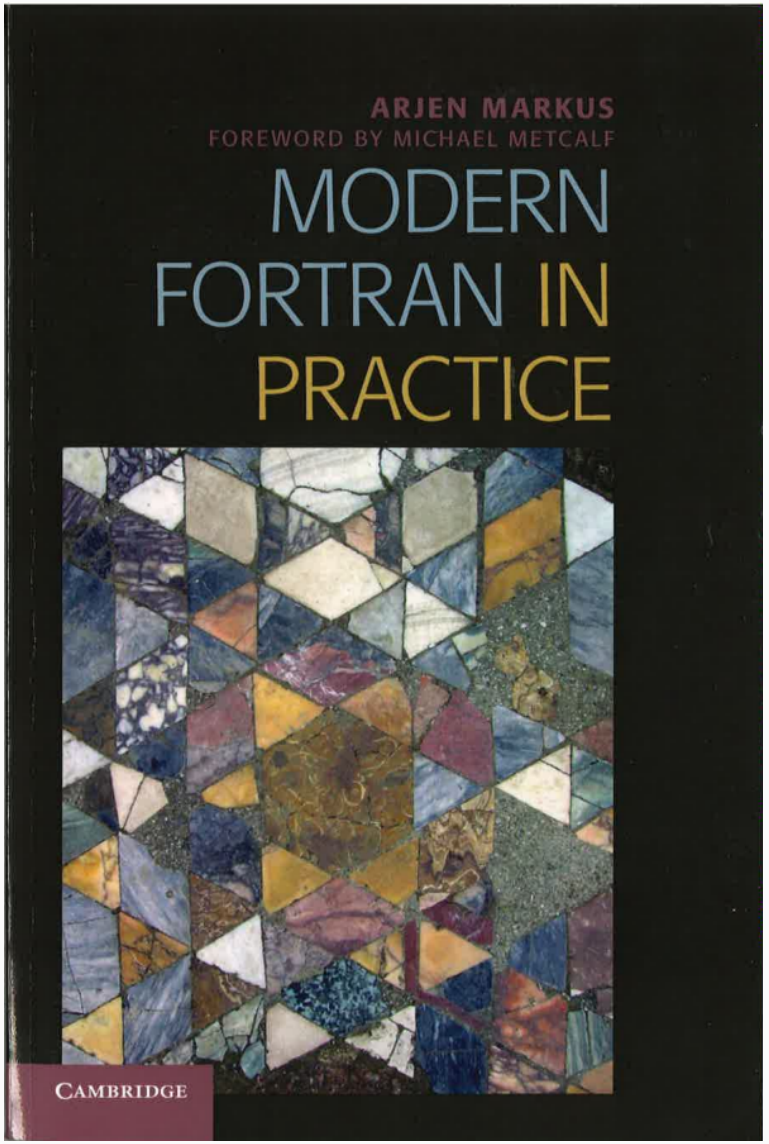
→ Use OO Polymorphism to pass function objects with common interfaces

Advantages

- No global data
- Decoupling nested function parameters from envelop function
- Smaller, modular functions
- No intermediate data storage
- Range-based functions possible
- Optimize one-element access

Literature


NORMAN S. CLERMAN • WALTER SPECTOR
Modern Fortran
STYLE AND USAGE
CAMBRIDGE


ARJEN MARKUS
FOREWORD BY MICHAEL METCALF
MODERN FORTRAN IN PRACTICE
CAMBRIDGE