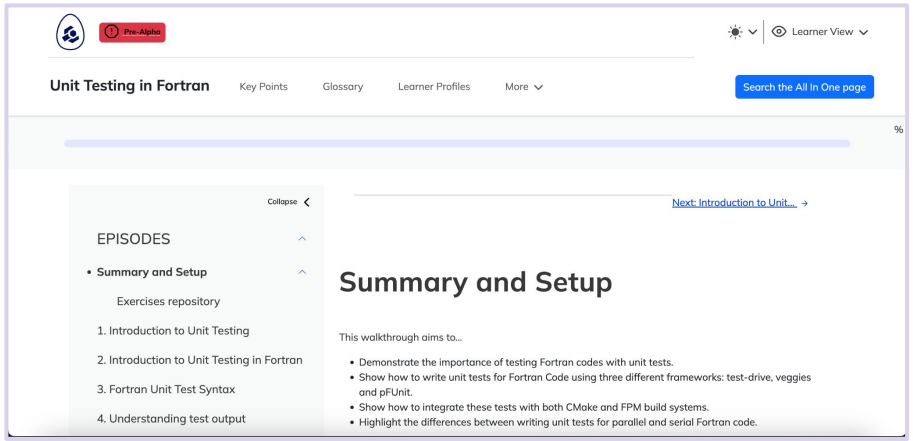
Unit Testing in Fortran

Connor Aird
RSE, Advanced Research Computing Centre (ARC)
UCL
c.aird@ucl.ac.uk

The Source Material

- Fortran unit testing lesson in the style of the software carpentries
- GitHub repository of exercises to act as challenges within lesson



THE CARPENTRIES

What is a Unit Test?

A way of verifying the validity of a code base by testing its smallest individual components, or **units**.

Unit tests are…

- **Isolated** - Do not rely on any other unit of code within the repository.
- **Minimal** - Test only one unit of code.
- **Fast** - Run on the scale of ms or s.

Why is it hard to unit test Fortran?

Fortran code can often have a lot of…

- Global variables
- Large, multipurpose procedures

The further we can move away from these practices, the easier it will be to unit test Fortran code

❌ **BAD →**

```
!> Evolve the board into the state of the next iteration
subroutine evolve_board()
   integer :: row, col, sum

   do row=2, nrow-1
       do col=2, ncol-1
           sum = 0
           sum = current_board(row, col-1)   &
               + current_board(row+1, col-1) &
               + current_board(row+1, col)   &
               + current_board(row+1, col+1) &
               + current_board(row, col+1)   &
               + current_board(row-1, col+1) &
               + current_board(row-1, col)   &
               + current_board(row-1, col-1)
           if(current_board(row,col)==1 .and. sum<=1) then
               new_board(row,col) = 0
           elseif(current_board(row,col)==1 .and. sum<=3) then
               new_board(row,col) = 1
           elseif(current_board(row,col)==1 .and. sum>=4)then
               new_board(row,col) = 0
           elseif(current_board(row,col)==0 .and. sum==3)then
               new_board(row,col) = 1
           endif
       enddo
   enddo

   return
end subroutine evolve_board
```

Why is it hard to unit test Fortran?

Fortran code can often have a lot of…

- Global variables
- Large, multipurpose procedures

The further we can move away from these practices, the easier it will be to unit test Fortran code

✅ **GOOD →**

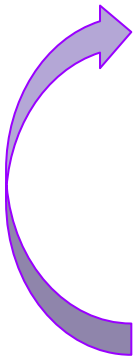
```
!> Evolve the board into the state of the next iteration
subroutine evolve_board(current_board, new_board)
   integer, dimension(:,:), allocatable, intent(in) :: current_board
   integer, dimension(:,:), allocatable, intent(inout) :: new_board

   integer :: row, col, num_rows, num_cols, sum

   num_rows = size(current_board, 1)
   num_cols = size(current_board, 2)

   do row=2, num_rows-1
       do col=2, num_cols-1
           sum = 0
           sum = current_board(row, col-1)   &
               + current_board(row+1, col-1) &
               + current_board(row+1, col)   &
               + current_board(row+1, col+1) &
               + current_board(row, col+1)   &
               + current_board(row-1, col+1) &
               + current_board(row-1, col)   &
               + current_board(row-1, col-1)
           if(current_board(row,col)==1 .and. sum<=1) then
               new_board(row,col) = 0
           elseif(current_board(row,col)==1 .and. sum<=3) then
               new_board(row,col) = 1
           elseif(current_board(row,col)==1 .and. sum>=4)then
               new_board(row,col) = 0
           elseif(current_board(row,col)==0 .and. sum==3)then
               new_board(row,col) = 1
           endif
       enddo
   enddo
end subroutine evolve_board
```

What test frameworks are there?

	pFUnit	test-drive	Veggies
CMake	✅	✅	✅
Fortran Package Manager (FPM)	❌	✅	✅
MPI	✅	❌	❌
OpenMP	❌	❌	❌
Array assertions	✅	❌	✅

What test frameworks are there?

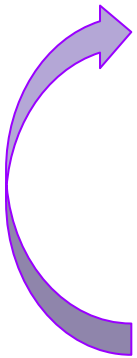
	pFUnit	test-drive	Veggies
CMake	✅	✅	✅
Fortran Package Manager (FPM)	❌	✅	✅
MPI	✅	❌	❌
OpenMP	❌	❌	❌
Array assertions	✅	❌	✅



Able to **parameterize** tests
using numbers of **MPI ranks**

What test frameworks are there?

	pFUnit	test-drive	Veggies
CMake	✅	✅	✅
Fortran Package Manager (FPM)	❌	✅	✅
MPI	✅	❌	❌
OpenMP	❌	❌	❌
Array assertions	✅	❌	✅



Able to **parameterize** tests
using numbers of **MPI ranks**

Writing a Serial Unit Test

How do you write a Fortran unit test?

Each testing framework follows a similar pattern…

```
module test_something
    ! use funit|pfunit|testdrive|veggies|
    ! use the src to be tested
    implicit none

    ! Define types to act as test parameters (and test case for pfunit)
contains

    ! Define a test suite (collection of tests) to be returned from a procedure

    ! Define the actual test execution code which will call the src and execute assertions

    ! Define constructors for your derived types (test parameters/cases)
end module test_something
```

How do you write a Fortran unit test?

Live Demo

Writing a serial unit test

episodes/3-fortran-unit-test-syntax/challenge

pFUnit task 2

How do you write a Fortran unit test?

Create test module - **test_find_steady_state.pf**

```
module test_find_steady_state
    use game_of_life_mod, only : find_steady_state        ! <-- Import the src to be tested
    use funit                                             ! <-- Import the serial pFUnit lib

    implicit none

    ! Define types to act as test parameters (and test case for pfunit)
contains

    ! Define a test suite (collection of tests) to be returned from a procedure

    ! Define the actual test execution code which will call the src and execute assertions

    ! Define constructors for your derived types (test parameters/cases)
end module test_find_steady_state
```

How do you write a Fortran unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
type, extends(AbstractTestParameter) :: find_steady_state_test_params ! <-- pFUnit type
```

```
end type find_steady_state_test_params
```

How do you write a Fortran unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
type, extends(AbstractTestParameter) :: find_steady_state_test_params
   !> The initial starting board to be passed into find_steady_state  ! <-- Inputs
   integer, dimension(:,:), allocatable :: input_board                ! <--
```

```
end type find_steady_state_test_params
```

How do you write a Fortran unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
type, extends(AbstractTestParameter) :: find_steady_state_test_params
   !> The initial starting board to be passed into find_steady_state
   integer, dimension(:,:), allocatable :: input_board
   !> The expected value of steady_state                    ! <-- Expected outputs
   logical :: expected_steady_state                         ! <--
   !> The expected output generation number                 ! <--
   integer :: expected_generation_number                    ! <--
```

```
end type find_steady_state_test_params
```

How do you write a Fortran unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
type, extends(AbstractTestParameter) :: find_steady_state_test_params
   !> The initial starting board to be passed into find_steady_state
   integer, dimension(:,:), allocatable :: input_board
   !> The expected value of steady_state
   logical :: expected_steady_state
   !> The expected output generation number
   integer :: expected_generation_number
   !> A description of the test to be outputted for logging          ! <-- Required for
   character(len=100) :: description                                ! <-- logging
Contains                                                            ! <--
   procedure :: toString => find_steady_state_test_params_toString  ! <--
end type find_steady_state_test_params
```

How do you write a Fortran unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
@testParameter                                                    ! <-- pFUnit macro
type, extends(AbstractTestParameter) :: find_steady_state_test_params
   !> The initial starting board to be passed into find_steady_state
   integer, dimension(:,:), allocatable :: input_board
   !> The expected value of steady_state
   logical :: expected_steady_state
   !> The expected output generation number
   integer :: expected_generation_number
   !> A description of the test to be outputted for logging
   character(len=100) :: description
contains
   procedure :: toString => find_steady_state_test_params_toString
end type find_steady_state_test_params
```

How do you write a Fortran unit test?

Define types to act as test parameters (and **test case** for pfunit)

```
type, extends(ParameterizedTestCase) :: find_steady_state_test_case    ! <-- pFUnit type

end type find_steady_state_test_case
```

How do you write a Fortran unit test?

Define types to act as test parameters (and **test case** for pfunit)

```
type, extends(ParameterizedTestCase) :: find_steady_state_test_case
   type(find_steady_state_test_params) :: params                    ! <-- Parameterised
end type find_steady_state_test_case
```

How do you write a Fortran unit test?

Define types to act as test parameters (and **test case** for pfunit)

```
@TestCase(testParameters={getTestSuite()}, constructor=paramsToCase)    ! <-- pFUnit macro
type, extends(ParameterizedTestCase) :: find_steady_state_test_case
   type(find_steady_state_test_params) :: params
end type find_steady_state_test_case
```

How do you write a Fortran unit test?

Define your testsuite (parameters)

```
function getTestSuite() result(params)
   type(find_steady_state_test_params), allocatable :: params(:)     ! <-- Returns a list of parameters
```

```
end function getTestSuite
```

How do you write a Fortran unit test?

Define your testsuite (parameters)

```
function getTestSuite() result(params)
   type(find_steady_state_test_params), allocatable :: params(:)

   integer, dimension(:,:), allocatable :: board                    ! <-- Populate the board for
                                                                    ! <-- each test case.
   allocate(board(31,31))                                           ! <-- In this scenario there is
   board = 0                                                        ! <-- only one
   board(9, 9:11) = [0,1,0]                                         ! <--
   board(10,9:11) = [1,1,1]                                         ! <--
   board(11,9:11) = [1,0,1]                                         ! <--
   board(12,9:11) = [0,1,0]                                         ! <--



end function getTestSuite
```

How do you write a Fortran unit test?

Define your testsuite (parameters)

```
function getTestSuite() result(params)
   type(find_steady_state_test_params), allocatable :: params(:)

   integer, dimension(:,:), allocatable :: board

   allocate(board(31,31))
   board = 0
   board(9, 9:11) = [0,1,0]
   board(10,9:11) = [1,1,1]
   board(11,9:11) = [1,0,1]
   board(12,9:11) = [0,1,0]
   params = [ &                                                    ! <--↓↓↓↓↓↓ Create the test cases
      find_steady_state_test_params(board, .true., 17, "an exploder initial state")]
end function getTestSuite
```

How do you write a Fortran unit test?

Define the actual test execution code which will call the src and execute assertions

```
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this              ! <-- Input the test case itself




end subroutine TestFindSteadyState
```

How do you write a Fortran unit test?

Define the actual test execution code which will call the src and execute assertions

```
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this

   logical :: actual_steady_state                                    ! <-- Define values to be checked
   integer :: actual_generation_number                               ! <-- and call src under test
                                                                     ! ↓↓↓↓↓↓↓↓↓↓↓↓
   call find_steady_state(.false., this%params%input_board, actual_steady_state, actual_generation_number)




end subroutine TestFindSteadyState
```

How do you write a Fortran unit test?

Define the actual test execution code which will call the src and execute assertions

```
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this

   logical :: actual_steady_state
   integer :: actual_generation_number

   call find_steady_state(.false., this%params%input_board, actual_steady_state, actual_generation_number)
                                                                    ! ↓↓↓↓↓↓ Check generation_number value
   @assertEqual(this%params%expected_generation_number, actual_generation_number, "Unexpected generation_number")



end subroutine TestFindSteadyState
```

How do you write a Fortran unit test?

Define the actual test execution code which will call the src and execute assertions

```
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this

   logical :: actual_steady_state
   integer :: actual_generation_number

   call find_steady_state(.false., this%params%input_board, actual_steady_state, actual_generation_number)

   @assertEqual(this%params%expected_generation_number, actual_generation_number, "Unexpected generation_number")
                                                                  ! ↓↓↓↓↓↓ Check steady_state value
   @assertTrue(this%params%expected_steady_state .eqv. actual_steady_state, "Unexpected steady_state value")

end subroutine TestFindSteadyState
```

How do you write a Fortran unit test?

Define the actual test execution code which will call the src and execute assertions

```
@Test                                                                    ! <-- pFUnit macro
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this

   logical :: actual_steady_state
   integer :: actual_generation_number

   call find_steady_state(.false., this%params%input_board, actual_steady_state, actual_generation_number)

   @assertEqual(this%params%expected_generation_number, actual_generation_number, "Unexpected generation_number")

   @assertTrue(this%params%expected_steady_state .eqv. actual_steady_state, "Unexpected steady_state value")

end subroutine TestFindSteadyState
```

How do you write a Fortran unit test?

Define constructors for your derived types (test **cases**/parameters)

```
function paramsToCase(testParameter) result(tst)                    ! <-- Convert params
   type(find_steady_state_test_params), intent(in) :: testParameter ! <-- to case
   type(find_steady_state_test_case) :: tst                          ! <--

end function paramsToCase
```

How do you write a Fortran unit test?

Define constructors for your derived types (test **cases**/parameters)

```
function paramsToCase(testParameter) result(tst)
   type(find_steady_state_test_params), intent(in) :: testParameter
   type(find_steady_state_test_case) :: tst
   tst%params = testParameter                                    ! <-- Copy params
end function paramsToCase
```

How do you write a Fortran unit test?

Define constructors for your derived types (test cases/**parameters**)

```
function find_steady_state_test_params_toString(this) result(string) ! <-- Convert params
   class (find_steady_state_test_params), intent(in) :: this          ! <-- to string
   character(:), allocatable :: string                                ! <--
```

```
end function find_steady_state_test_params_toString
```

How do you write a Fortran unit test?

Define constructors for your derived types (test cases/**parameters**)

```
function find_steady_state_test_params_toString(this) result(string)
   class (find_steady_state_test_params), intent(in) :: this
   character(:), allocatable :: string

   character(len=80) :: buffer                                    ! <-- Populate a buffer with
   integer :: nrow, ncol                                          ! <-- some text to be
                                                                  ! <-- logged during testing
   nrow = size(this%input_board, 1)                               ! <--
   ncol = size(this%input_board, 2)                               ! <--
   write(buffer,'(i2, "x", i2, " board with ", a)') &             ! <--
       nrow, ncol, trim(this%description)                         ! <--


end function find_steady_state_test_params_toString
```

How do you write a Fortran unit test?

Define constructors for your derived types (test cases/**parameters**)

```
function find_steady_state_test_params_toString(this) result(string)
   class (find_steady_state_test_params), intent(in) :: this
   character(:), allocatable :: string

   character(len=80) :: buffer
   integer :: nrow, ncol

   nrow = size(this%input_board, 1)
   ncol = size(this%input_board, 2)
   write(buffer,'(i2, "x", i2, " board with ", a)') &
       nrow, ncol, trim(this%description)

   string = trim(buffer)                                    ! <-- Save the buffer
end function find_steady_state_test_params_toString
```

Integrating with build systems

pFUnit CMake configuration

```
find_package(PFUNIT REQUIRED)                    # <-- Find pFUnit lib from CMAKE_PREIX_PATH
```

Integrating with build systems

pFUnit CMake configuration

```
find_package(PFUNIT REQUIRED)
enable_testing()                                    # <-- Enable ctest
```

Integrating with build systems

pFUnit CMake configuration

```
find_package(PFUNIT REQUIRED)
enable_testing()

add_library (sut STATIC ${PROJ_SRC_FILES})                    # <-- Create a src library
```

Integrating with build systems

pFUnit CMake configuration

```
find_package(PFUNIT REQUIRED)
enable_testing()

add_library (sut STATIC ${PROJ_SRC_FILES})

file(GLOB test_srcs "${PROJECT_SOURCE_DIR}/test/pfunit/*.pf") # <-- Filter all test files to just the
                                                              # <-- find_steady_state test
set(test_find_steady_state_src ${test_srcs})                  # <--
list(FILTER test_find_steady_state_src                        # <--
     INCLUDE REGEX ".*test_find_steady_state.pf")             # <--
```

Integrating with build systems

pFUnit CMake configuration

```
find_package(PFUNIT REQUIRED)
enable_testing()

add_library (sut STATIC ${PROJ_SRC_FILES})

file(GLOB test_srcs "${PROJECT_SOURCE_DIR}/test/pfunit/*.pf")

set(test_find_steady_state_src ${test_srcs})
list(FILTER test_find_steady_state_src
     INCLUDE REGEX ".*test_find_steady_state.pf")

add_pfunit_ctest (pfunit_find_steady_state_tests          # <-- Add test to ctest with the
 TEST_SOURCES ${test_find_steady_state_src}               # <-- provided pfunit function
 LINK_LIBRARIES sut)                                     # <--
```

Writing a Parallel Unit test

What do we want from a parallel unit test?

There are a few key things we need a parallel unit test to handle…

- Running with different numbers of MPI ranks for a single mpirun execution.
- Asserting different things for different ranks within the same test case.

How do you write a parallel unit test?

Live Demo

Writing a parallel unit test

episodes/5-testing-parallel-code/challenge

How do you write a parallel unit test?

Use **pfunit** instead of funit

```
module test_find_steady_state
    use game_of_life_mod, only : find_steady_state     ! <-- Import the src to be tested
    use pfunit                                         ! <-- Import the parallel pFUnit lib

    implicit none

    ! Define types to act as test parameters (and test case for pfunit)
contains

    ! Define a test suite (collection of tests) to be returned from a procedure

    ! Define the actual test execution code which will call the src and execute assertions

    ! Define constructors for your derived types (test parameters/cases)
end module test_find_steady_state
```

How do you write a parallel unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
type, extends(MPITestParameter) :: find_steady_state_test_params      ! <-- Extend MPITestParameter
```

```
end type find_steady_state_test_params
```

How do you write a parallel unit test?

Define types to act as **test parameters** (and test case for pfunit)

```
@testParameter                                                    ! <-- No change
type, extends(MPITestParameter) :: find_steady_state_test_params
   !> The initial starting board to be passed into find_steady_state  ! <-- No change
   integer, dimension(:,:), allocatable :: input_board            ! <--
   !> The expected steady state result                             ! <--
   logical :: expected_steady_state                               ! <--
   !> The expected number of generations to reach steady state     ! <--
   integer :: expected_generation_number                          ! <--
   !> A description of the test to be outputted for logging        ! <--
   character(len=100) :: description                              ! <--
contains                                                          ! <--
   procedure :: toString => find_steady_state_test_params_toString  ! <--
end type find_steady_state_test_params
```

How do you write a parallel unit test?

Define types to act as test parameters (and **test case** for pfunit)

```
type, extends(MPITestCase) :: find_steady_state_test_case          ! <-- Extend MPITestCase

end type find_steady_state_test_case
```

How do you write a parallel unit test?

Define types to act as test parameters (and **test case** for pfunit)

```
@TestCase(testParameters={getTestSuite()}, constructor=paramsToCase) ! <-- No change
type, extends(MPITestCase) :: find_steady_state_test_case
   type(find_steady_state_test_params) :: params                     ! <-- No change
end type find_steady_state_test_case
```

How do you write a parallel unit test?

Set the number of MPI ranks for each test case

```
function getTestSuite() result(params)                          ! <-- No change to the signature or
   type(find_steady_state_test_params), allocatable :: params(:)  ! <-- how we populate the board
                                                                  ! <--
                                                                  ! <--
   integer, dimension(:,:), allocatable :: board                  ! <--
                                                                  ! <--
   allocate(board(31, 31))                                        ! <--
   board = 0                                                      ! <--
   board(9,9:11)  = [0,1,0]                                       ! <--
   board(10,9:11) = [1,1,1]                                       ! <--
   board(11,9:11) = [1,0,1]                                       ! <--
   board(12,9:11) = [0,1,0]                                       ! <--
```

```
end function getTestSuite
```

How do you write a parallel unit test?

Set the number of MPI ranks for each test case

```
function getTestSuite() result(params)
   type(find_steady_state_test_params), allocatable :: params(:)

   integer :: i, max_num_ranks = 8                                  ! <-- Additional variables required
   integer, dimension(:,:), allocatable :: board

   allocate(board(31, 31))
   board = 0
   board(9,9:11)  = [0,1,0]
   board(10,9:11) = [1,1,1]
   board(11,9:11) = [1,0,1]
   board(12,9:11) = [0,1,0]

   allocate(params(max_num_ranks))                                  ! <-- Add a set of parameters for each number of ranks
   do i = 1, max_num_ranks                                          ! <--↓↓↓↓↓↓
       params(i) = find_steady_state_test_params(i, board, .true., 17, "an exploder initial state")
   end do
end function getTestSuite
```

How do you write a parallel unit test?

Update the call to find steady state

```
@Test                                                                    ! <-- No change
subroutine TestFindSteadyState(this)                                     ! <--
   class(find_steady_state_test_case), intent(inout) :: this             ! <--
                                                                         ! <--
   logical :: actual_steady_state                                        ! <--
   integer :: actual_generation_number                                   ! <--









                                                                         ! ↓↓↓↓↓↓ No change
   @assertEqual(this%params%expected_generation_number, actual_generation_number, "Unexpected generation_number")
   @assertTrue(this%params%expected_steady_state .eqv. actual_steady_state, "Unexpected steady_state value")


end subroutine TestFindSteadyState
```

How do you write a parallel unit test?

Update the call to find steady state

```
@Test
subroutine TestFindSteadyState(this)
   class(find_steady_state_test_case), intent(inout) :: this

   logical :: actual_steady_state
   integer :: actual_generation_number
                                                          ! ↓↓↓↓↓↓ Use new signature
   call find_steady_state(actual_steady_state, actual_generation_number, this%params%input_board, &
          size(this%params%input_board, 1), size(this%params%input_board, 2), &
          this%getMpiCommunicator(), this%getNumProcessesRequested())

   @assertEqual(this%params%expected_generation_number, actual_generation_number, "Unexpected generation_number")
   @assertTrue(this%params%expected_steady_state .eqv. actual_steady_state, "Unexpected steady_state value")

end subroutine TestFindSteadyState
```

How do you write a parallel unit test?

Update CMakeLists.txt to mark the test as parallel

```
find_package(PFUNIT REQUIRED)                                    # <-- No change
enable_testing()                                                 # <--
                                                                 # <--
add_library (sut STATIC ${PROJ_SRC_FILES})                       # <--
                                                                 # <--
file(GLOB test_srcs "${PROJECT_SOURCE_DIR}/test/pfunit/*.pf")    # <--
                                                                 # <--
set(test_find_steady_state_src ${test_srcs})                     # <--
list(FILTER test_find_steady_state_src                           # <--
     INCLUDE REGEX ".*test_find_steady_state.pf")                # <--
```

How do you write a parallel unit test?

Update CMakeLists.txt to mark the test as parallel

```
find_package(PFUNIT REQUIRED)
enable_testing()

add_library (sut STATIC ${PROJ_SRC_FILES})

file(GLOB test_srcs "${PROJECT_SOURCE_DIR}/test/pfunit/*.pf")

set(test_find_steady_state_src ${test_srcs})
list(FILTER test_find_steady_state_src
     INCLUDE REGEX ".*test_find_steady_state.pf")

add_pfunit_ctest (pfunit_find_steady_state_tests          # <-- Specify a maximum number of MPI processors
 TEST_SOURCES ${test_find_steady_state_src}               # <--
 LINK_LIBRARIES sut                                      # <--
 MAX_PES 8)                                              # <-- This line
```

Tips for writing testable parallel code?

Some tips for writing parallel unit tests…

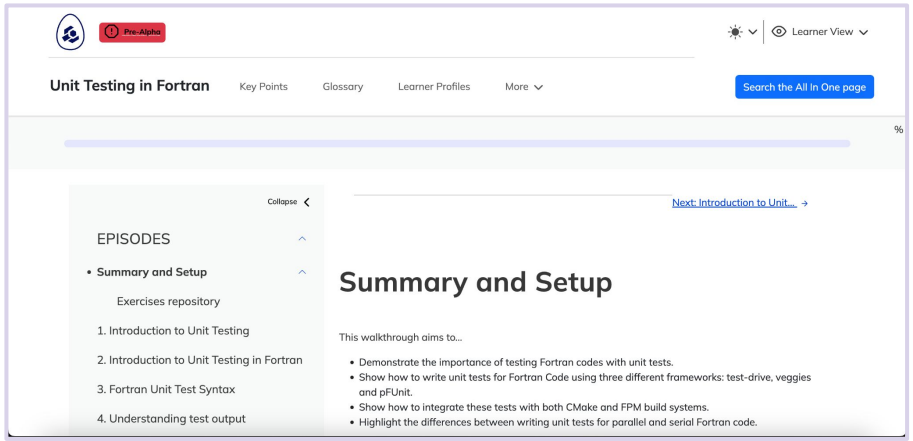
- Not all tests need to be parallel.
  - If a procedure does not call the MPI library it does not need to be tested in parallel.
- Isolate calls to the MPI library into procedures.
  - This allows testing more procedures using serial tests.
- Pass the MPI communicator into procedures which call the MPI library.
  - This allows the test library to set the communicator when testing.

Thank You

Connor Aird
RSE, Advanced Research Computing Centre (ARC)
UCL
c.aird@ucl.ac.uk

The Source Material

- Fortran unit testing lesson in the style of the software carpentries
- GitHub repository of exercises to act as challenges within lesson



THE CARPENTRIES

Try it yourself

Open a codespace in the exercises repository

github.com/UCL-ARC/fortran-unit-testing-exercises