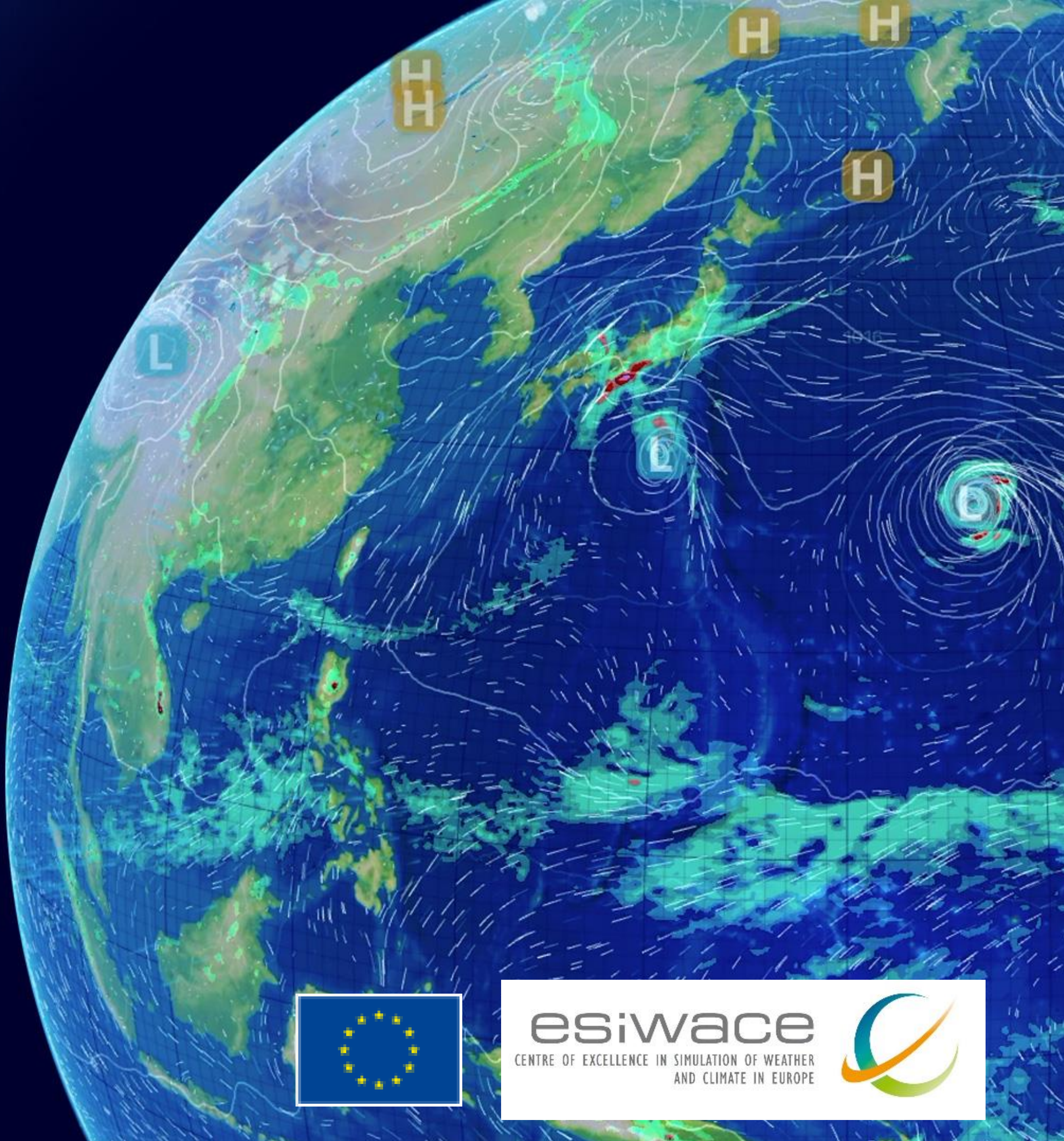
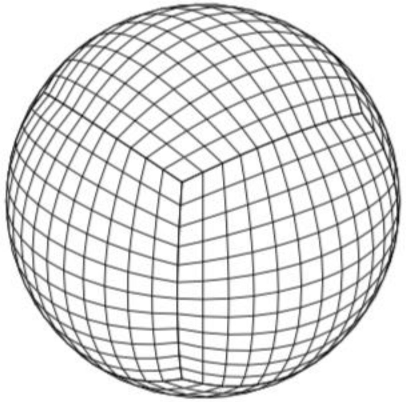
PSyclone in LFRic

Iva Kavčič, Met Office, UK &

Rupert Ford, Andrew Porter, Sergi Siso (STFC, UK); Joerg Henrichs (BOM, AU); Wolfgang Hayek (NIWA, NZ); Christopher Maynard, Ben Shipway (UKMO) + many others…

ESiWACE2 2nd Virtual Workshop on Emerging Technologies for Weather and Climate Modelling, 07 October 2020





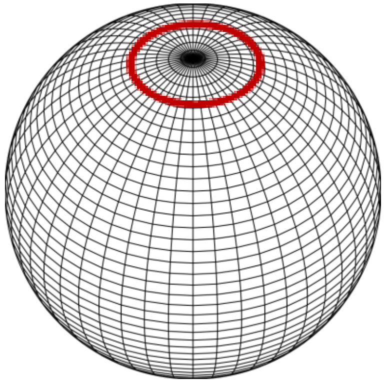
LFRic (*after **L**ewis **Fr**y **Ric**hardson*) is the new weather and climate modelling system being developed by the UK Met Office to replace the existing Unified Model in preparation for exascale computing in the 2020s

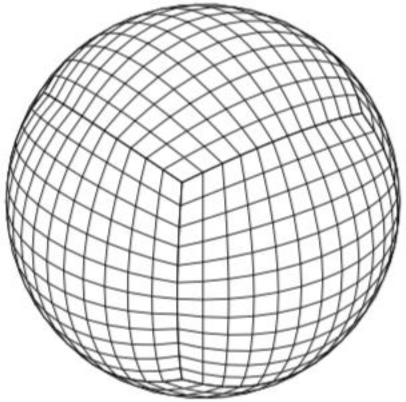
- Uses the **GungHo dynamical core**
- Runs on a **semi-structured cubed-sphere mesh**

PSyclone is a domain-specific compiler and source-to-source translator developed for use in finite element, finite volume and finite difference codes

- Uses the **information** from a supported **API**
- **Generates code** exploiting different **parallel programming models**

UM to LFRic: Optimisations





Unified Model (UM) & ENDGame dynamical core

- Fully structured Lat-Lon mesh
- Staggered Finite Differences (FDM)
- ***Hard-coded optimisations***

LFRic system & GungHo dynamical core

- Horizontally unstructured, vertically structured quasi-uniform mesh
- Mixed Finite Elements (FEM)
- ***Generated optimisations***

PSyclone

PSyclone 2.2.0
BSD 3-clause
https://github.com/stfc/PSyclone
https://psyclone.readthedocs.io
```
> pip install psyclone
```

- A **domain-specific compiler** for embedded DSL(s)
  - Configurable: FD/FV NEMO, GOcean, FE LFRic
  - Currently Fortran -> Fortran/OpenCL
  - Supports distributed- and shared-memory parallelism
  - Supports **code generation** and **code transformation**
- A tool for use by HPC experts
  - Hard to beat a human (debatable)
  - Work round limitations/bugs
  - Optimisations encoded as a 'recipe' rather than baked into the scientific source code
  - Different recipes for different computer architectures (CPU, GPU)
  - Enables scriptable, whole-code optimisation

What PSyclone does in LFRic (and how)

Developing PSyclone features for LFRic

Building LFRic with PSyclone

Management of PSyclone in Met Office

Met Office

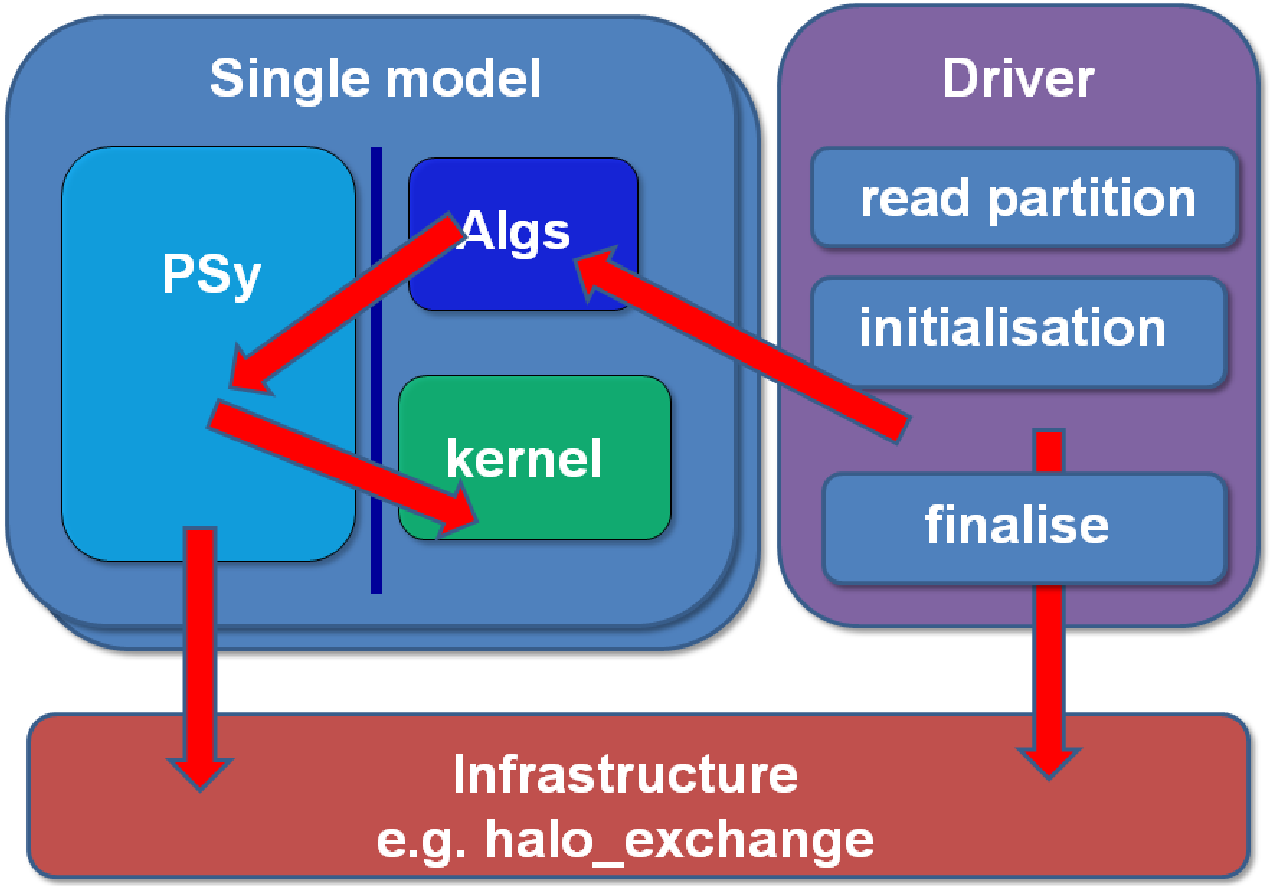
LFRic Separation of Concerns: Science and code optimisation

PSyKAl

- **Algorithms:** Natural Science, operations on whole **data structures** (e.g. fields)
- **Parallel-Systems:** Computational Science, applies optimisations (and unpacks data) – ***generated code***

PSyclone

- **Kernels:** Natural Science, operations on (columns of) data points



DSL embedded in Fortran: Algorithm code (operations on whole fields; *written by scientists, Fortran 2008)*

```
module rhs_rho_alg_mod
…
 subroutine on_the_fly_rhs_alg(rhs, state, ref_state, … )
   use rrho_kernel_mod,          only: rrho_kernel_type
   use matrix_vector_kernel_mod, only: matrix_vector_kernel_type
   implicit none
   type(field_type), target, intent(in) :: state(bundle_size)
   type(field_type), target, intent(inout) :: rhs(bundle_size)
   …
   call invoke( name = "compute_rhs_rho",                              &
                rtheta_kernel_type( rhs_tmp(igh_t), rho_ref, u, u_ref ), &
                matrix_vector_kernel_type( rhs(igh_t), theta, mm_rho ),  &
                inc_X_plus_bY( rhs(igh_t), tau_t_dt, rhs_tmp(igh_t) ) )
  …
  end subroutine on_the_fly_rhs_alg
end module rhs_rho_alg_mod
```

***Global fields:** data layout hidden*

Kernel (LFRic)

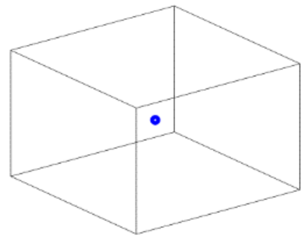
- PSy-layer loop over **columns of cells**

Built-in (PSyclone)

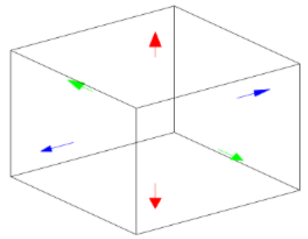
- PSy-layer loop over all field **DoFs** (arithmetic operations)

DSL embedded in Fortran: Kernel metadata (how to **access** and **update data**; *kernel code written by scientists, Fortran 90*)

```
module rrho_kernel_mod
…
  type, public, extends(kernel_type) :: rrho_kernel_type
    private
    type(arg_type) :: meta_args(4) = (/                            &
         arg_type(GH_FIELD, GH_REAL, GH_READWRITE, W3),            &
         arg_type(GH_FIELD, GH_REAL, GH_READ,                      &
                                     ANY_DISCONTINUOUS_SPACE_1),   &
         arg_type(GH_FIELD, GH_REAL, GH_INC,       W2),            &
         arg_type(GH_FIELD, GH_REAL, GH_READ,      ANY_SPACE_1),   &
         /)
    integer :: operates_on = CELL_COLUMN
  contains
    procedure, nopass :: rrho_code
  end type
…
```

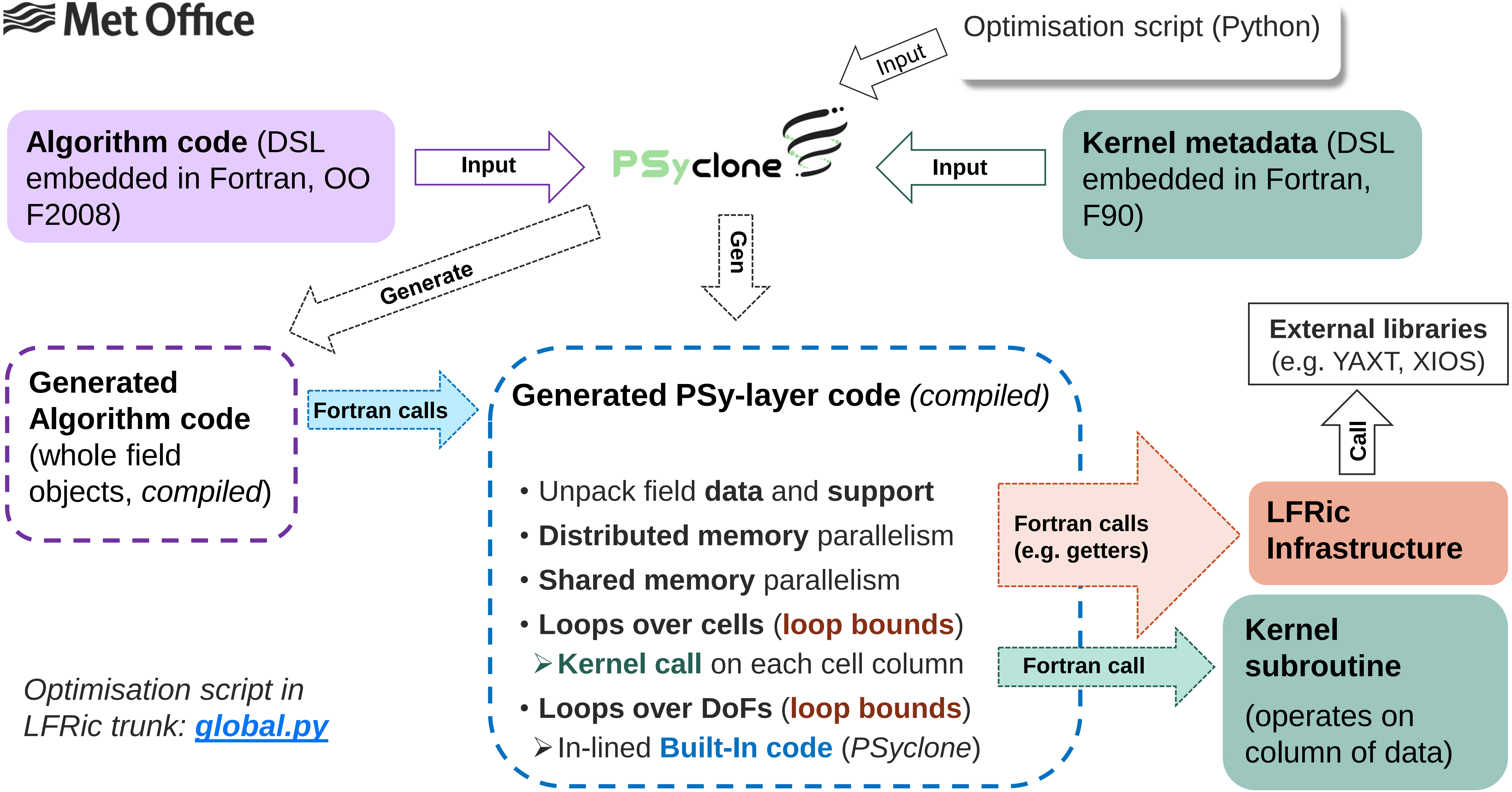


Discontinuous *function spaces: no shared DoFs*



Continuous *function spaces: shared DoFs (**colour** OpenMP loops)*

Met Office



*Optimisation script in LFRic trunk: **global.py***

Met Office

Rules of engagement: LFRic ⟷ PSyclone LFRic API

LFRic

- **Rules** (e.g. writing algorithms & kernels, data properties)
- Kernel **metadata**
- **Infrastructure support** (e.g. dofmap, colouring, MPI)

LFRic API

- Flexible **optimisations**
- Support for data structures & properties following **API rules**
- Development mandated by requirements

PSyKAl-lite

Required functionality (hand-written)

Distributed + shared memory parallelism

PSyclone transformations

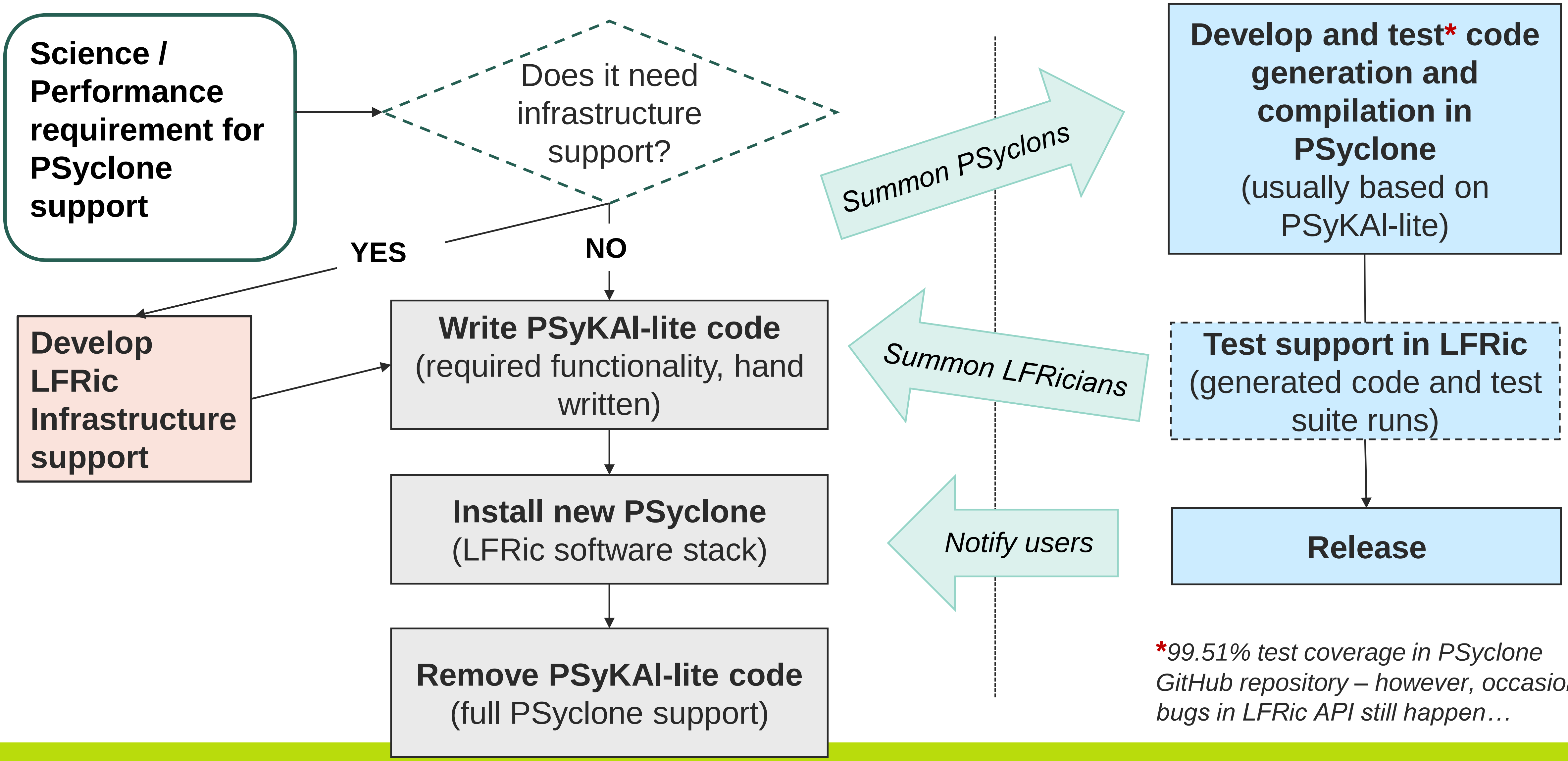
- Dynamo0p3ColourTrans
- Dynamo0p3OMPLoopTrans
- OMPParallelTrans
- Dynamo0p3RedundantComputationTrans

LFRic infrastructure support

- Halo exchange
- Colouring (cell & halo) for fields on continuous FS

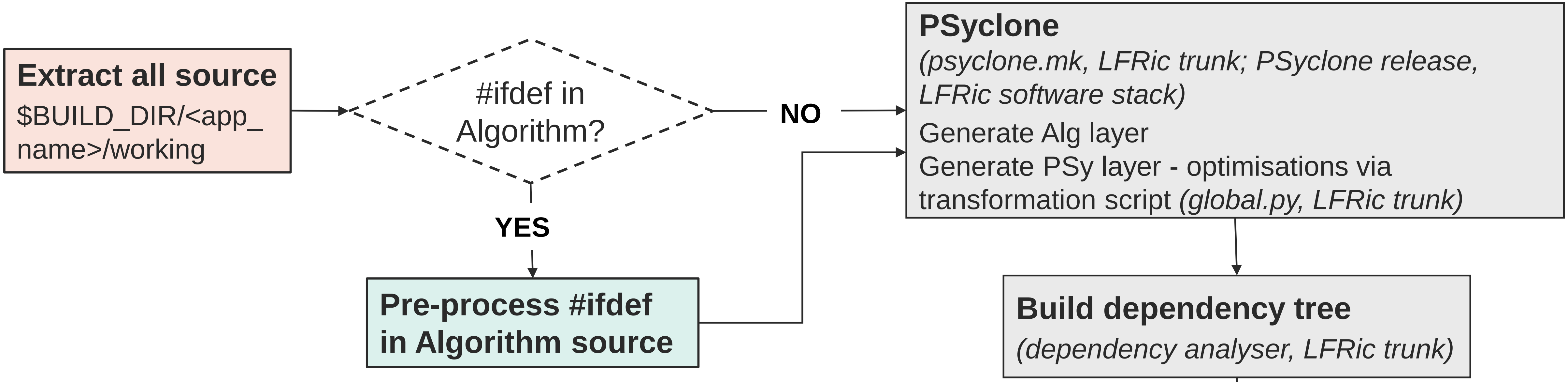
…

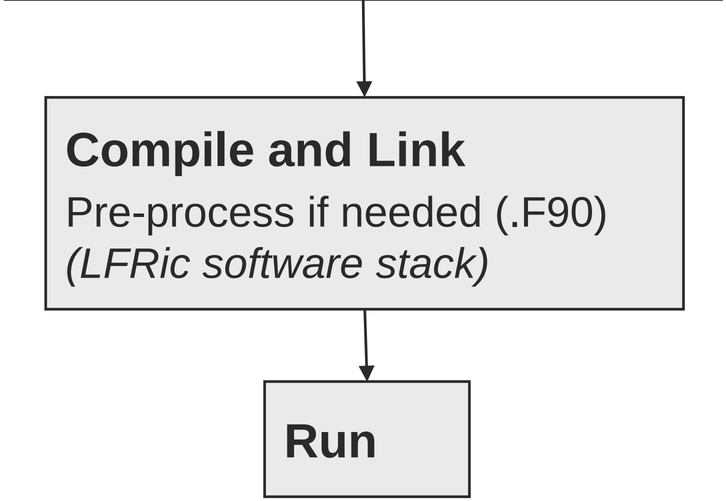
Development: LFRic ↔ PSyclone



*****_99.51% test coverage in PSyclone GitHub repository – however, occasional bugs in LFRic API still happen…_

Building LFRic with PSyclone





Naming convention for Alg and PSy source in LFRic build system

- **Alg source** (not compiled): <alg_name>_alg_mod.**x90**, .**X90**
- Generated Alg (compiled): <alg_name>_alg_mod.f90
- Generated PSy (compiled): <alg_name>_alg_mod_psy.f90

Management of PSyclone at Met Office: TODOs / Challenges

- Building **generated code** that is not in repository
  - **Fab build system** developed by the MO (intended for general use).
- Different projects/models will need **different PSyclone releases**
  - Management of PSyclone installations for different projects/models.
  - Ideally, this would be coordinated across NGMS Programme/MO – resources?
- **Porting** to Gen 1 and Gen 2 architecture
  - PSyclone releases with different **Python environments** *(LFRic builds its own Python virtual environments from AVD SciTools because of library clashes).*
  - Different releases / compiler environments / configurations?

Met Office

Management of PSyclone at Met Office: TODOs / Challenges

- **Configuring** PSyclone releases per API and/or model run
  - LFRic **modifies** PSyclone configuration file during the installation process.
  - Flexible builds will need to access different configuration files → work in progress on configurable runs (configuration stored in the LFRic repository).
- *Continuous testing of LFRic code against PSyclone master
  - Some bugs PSyclone LFRic API get missed in between releases due to rapid development of LFRic code *(usually quickly fixed due to* ***close collaboration between STFC and MO****).*
  - PSyclone test environments are currently manually upgraded as needed (installation scripts) to test LFRic trunk or branch → work in progress on automation.

Met Office Science & Technology Facilities Council esiwace CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Met Office

Highlights and development

Mixed precision and i-first looping in LFRic Atmosphere

Work on colouring algorithm

Mixed precision and i-first data layout in LFRic Atmosphere

- PSyclone and LFRic infrastructure support for **mixed precision** and fields/operator of **different datatypes**.

- LFRic utilises existing parameterisation schemes for the Unified model.
  - **Interfacing** (“Physics” kernels) **UM Physics**, **SOCRATES** ("Suite Of Community RAdiative Transfer codes") and **JULES** (land surface) models.
  - PSyclone/LFRic infrastructure support for *i-first* looping with entire fields passed to the interfacing kernels (“operate_on = DOMAIN”).

Get the data in the right place

- **Design and implementation of shared memory parallelism and i-first indexing for UM physics parametrisations**

GungHo dycore	UM physics
Same operation within a vertical column	Same operation within a horizontal layer
k-first array index	i-first array index
Good for cache re-use	Good for Single Instruction Multiple Data

Original Structure:

Psy layer:
Loop over horizontal domain
Call kernel on column

```
integer :: operates_on = CELL_COLUMN
```

Kernel:
Convert data to column of UM data
Call UM scheme on column
Convert data back to LFRic

New i-first Structure:

Psy layer:
Call kernel

```
integer :: operates_on = DOMAIN
```

Kernel:
Convert data to whole-domain, i-first, UM data
Call UM scheme on whole domain
Convert data back to LFRic

Courtesy of Ben Shipway, UKMO

Computational benefits

Single precision solver & i-first microphysics

Solver precision

Configuration	Trunk [33756]	Branch 64bit	Branch 32bit
6omp 6nodes	176.07	166.00	98.38
6omp 12nodes	86.05	78.21	55.57
6omp 24nodes	49.73	45.51	32.49

i-first microphysics

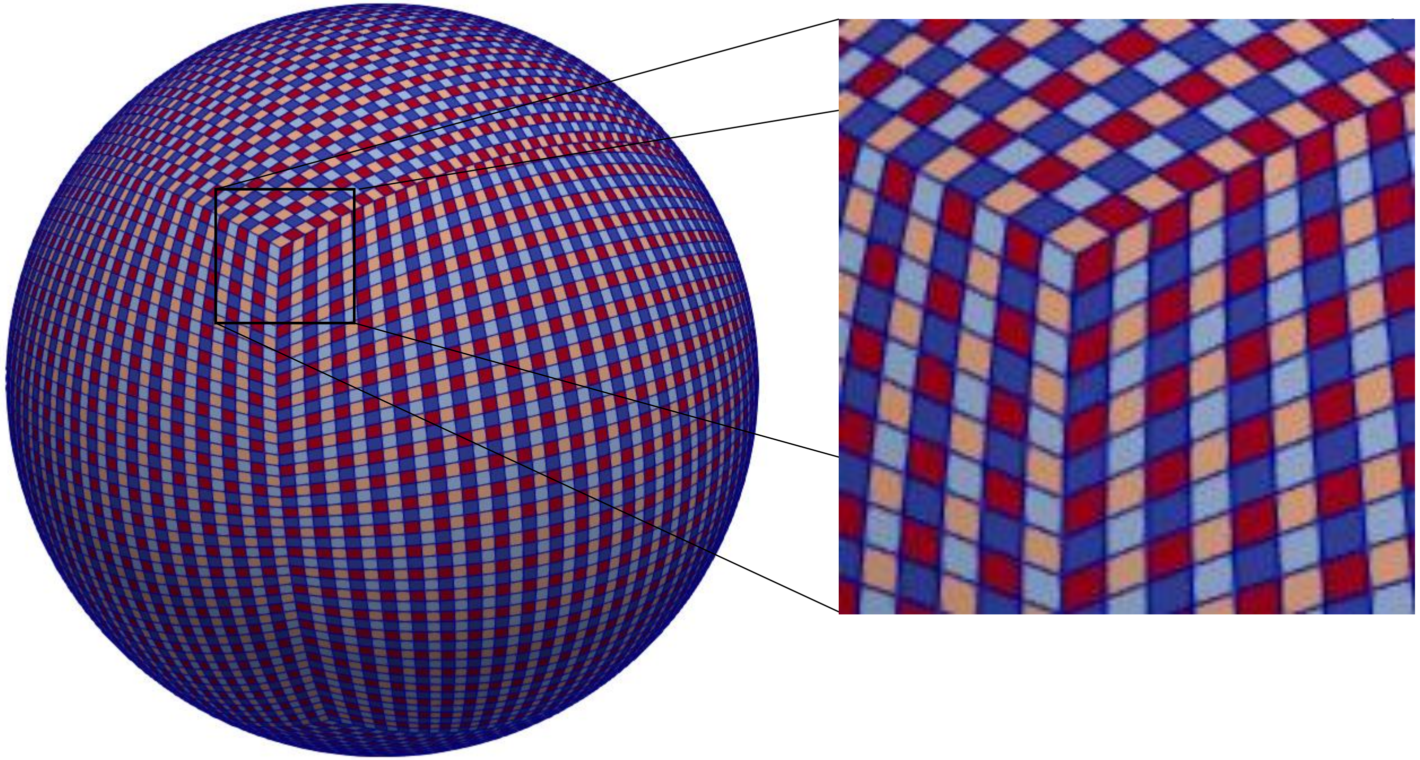
Microphysics timings	k-first (original)	i-first
C48, 216 PEs, 720 timesteps	247.4s	63.9s
C192, 864 PEs, 60 timesteps	33.5s	9.6s

Courtesy of Ben Shipway, UKMO

Impact of colouring algorithm on kernel performance (Wolfgang Hayek, NIWA)

- Exploit cubed-sphere structure to improve kernel performance.
- Improve cache reuse and memory bandwidth utilisation.
- Look at cell colouring and data column order in field arrays.
- Compare performance between (n x m) tile setups with tile-ordered colour maps.
- Simplifications: use 24 colours (4 per cube face) to avoid data races, no field array reordering yet.
- Use Chris Maynard' microbenchmark app and Sergi Siso's optimised matrix-vector kernel with vectorised k-inner loop.

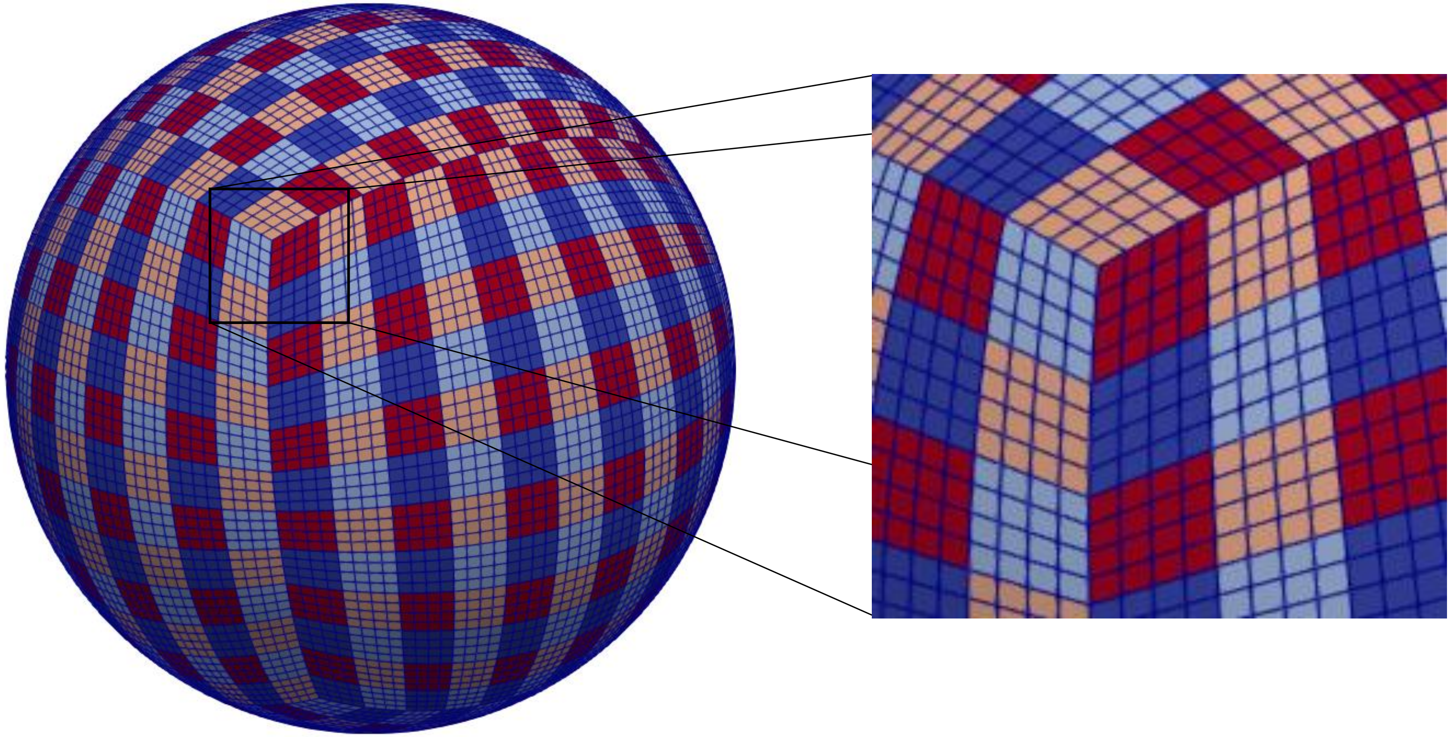
Current Scheme



- Uses 4 colours, staggered cell order in colour maps
- No cache-reuse for shared DOFs on cell vertices/edges/faces
- Data column order in field arrays follows mesh generator-provided order (?)

Courtesy of Wolfgang Hayek, NIWA

Coloured Tiles

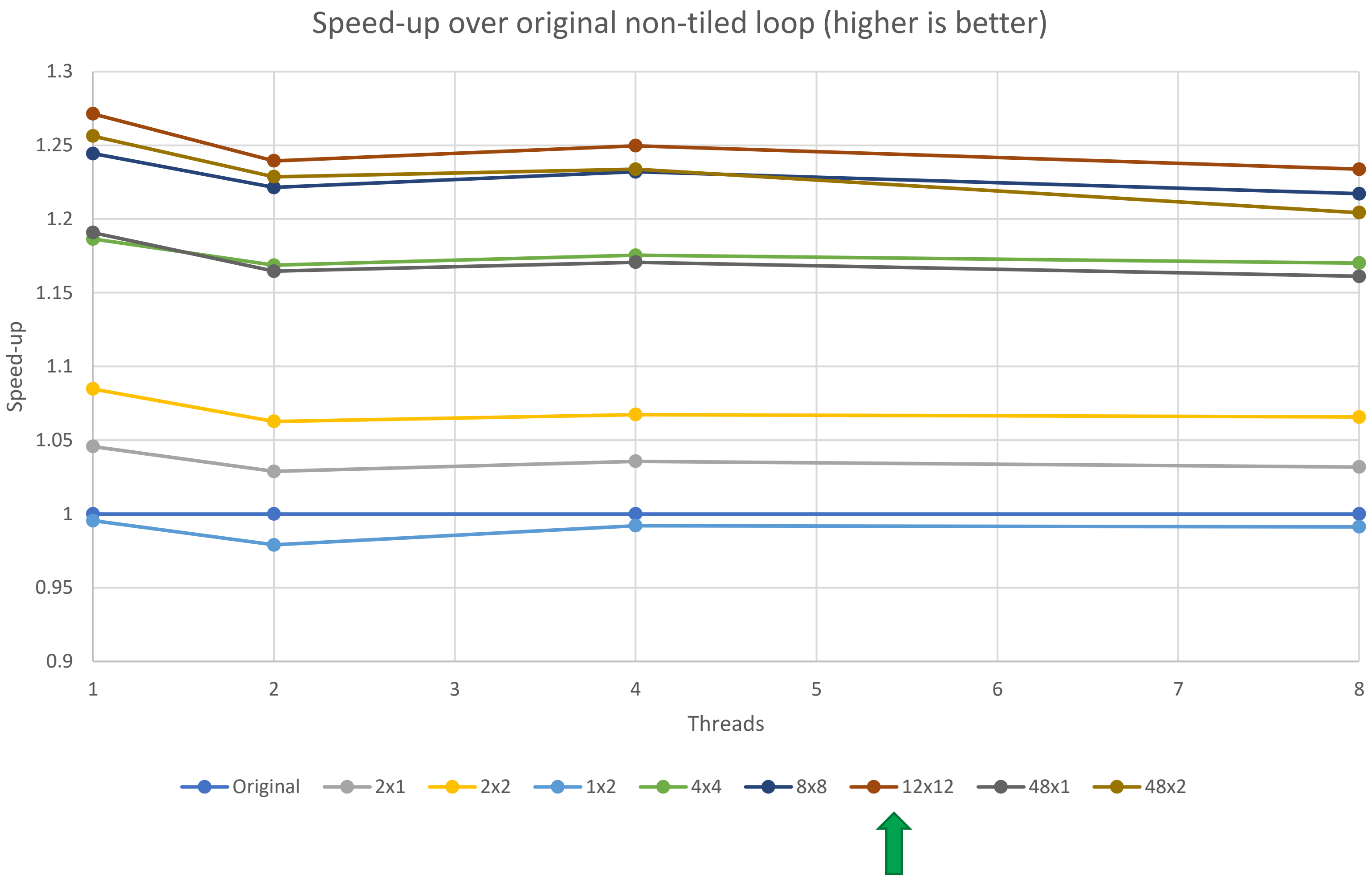


- Use coloured (n x m) tiles and reorder cell IDs tile-wise contiguously in colour maps
- Optimise data order in field arrays to follow tile structure
- Parallelise over tiles of the same colour to avoid data races
- Should work on CPUs and GPUs, and for domain-decomposed cubed-spheres and LAMs

Courtesy of Wolfgang Hayek, NIWA

Results

- *C48 cubed-sphere mesh, 120 vertical levels*
- *Build with Intel v18.0.1 compiler with "–O2 –qopenmp –qopenmp-simd"*

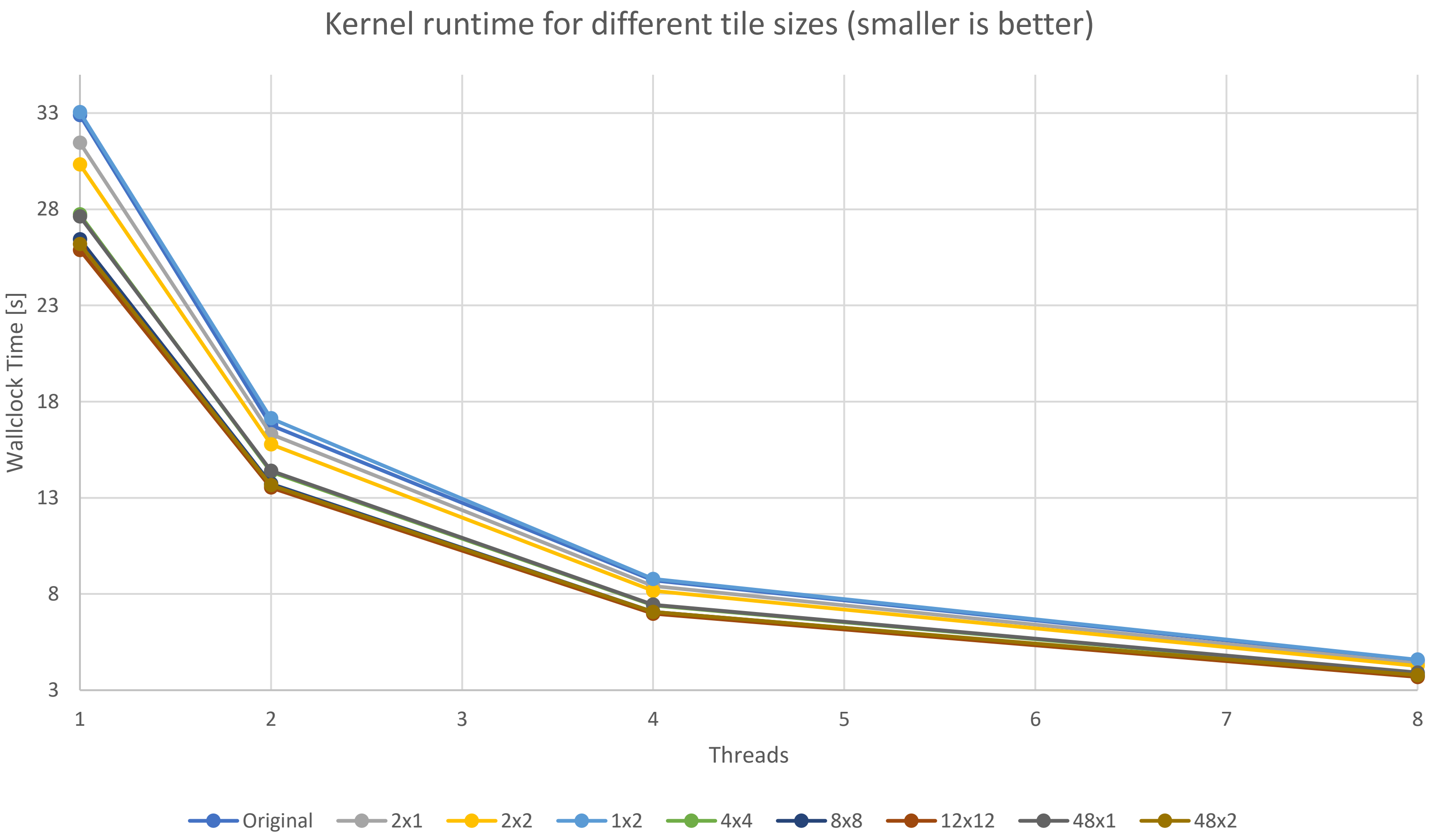


Run on Intel Xeon Gold 6148 2.4 GHz (Skylake), pin threads to physical cores in same socket, 1000 kernel iterations, best-of-5 timings

Up to x1.27 speed-up

Courtesy of Wolfgang Hayek, NIWA

Results



Performance improvement confirmed by reduced LLC miss rates (VTune)

Courtesy of Wolfgang Hayek, NIWA

Met Office

Thank you ☺

Acknowledgements

PSyclone developers, LFRic team, GungHo Atmospheric Science team and many others…

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988

Extras

Links and references

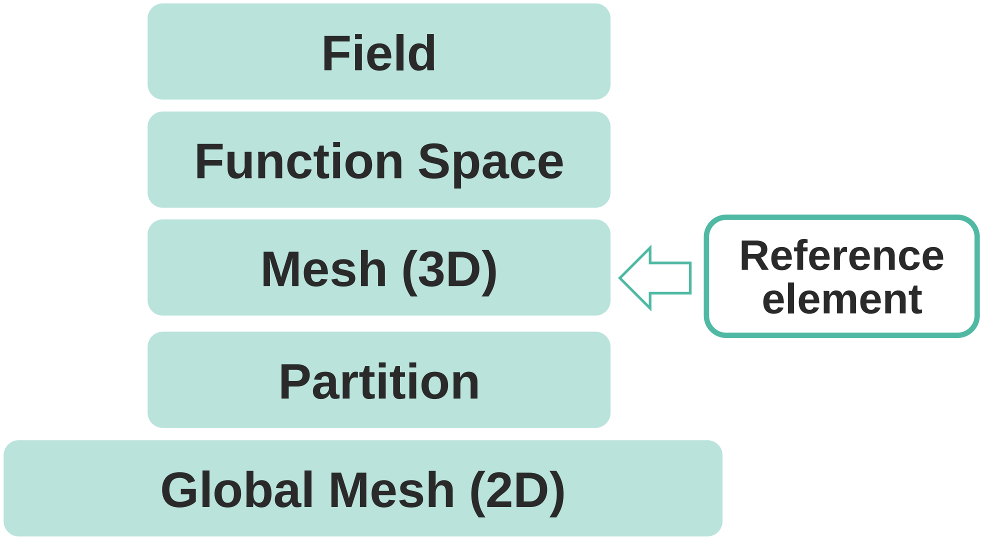
- PSyclone and fparser
  - https://github.com/stfc/PSyclone
  - https://psyclone.readthedocs.io
  - https://github.com/stfc/fparser
  - https://fparser.readthedocs.io
- LFRic repository (email Steve Mullerworth, LFRic Team manager, for access): https://code.metoffice.gov.uk/trac/lfric/wiki
- LFRic revision – PSyclone release compatibility: https://code.metoffice.gov.uk/trac/lfric/wiki/LFRicTechnical/VersionsCompatibility#LFRicCompatibility
- LFRic Singularity container: https://github.com/NCAS-CMS/LFRic_container
- LFRic software stack recipes: https://github.com/MetOffice/NGMS-SoftwareStack
- PSyclone in LFRic: https://code.metoffice.gov.uk/trac/lfric/wiki/PSycloneTool
- Stylist: https://github.com/MetOffice/stylist
- Adams et al. (2019), *LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models*, Journal of Parallel and Distributed Computing,132, 383-396

Infrastructure and data layout

LFRic Software Engineering: LFRic infrastructure (OO F2008)

- **LFRic: data classes**
  - Storing prognostic and diagnostic quantities: **field**;
  - Mathematical operations: **operator** (FEM matrices/FS mappings), **scalar** (global reductions).
- **Data classes** for supporting objects, e.g., mesh, reference element, function space.
- **Challenge: Compiler support for object-orientated F2008** is mixed → "compiler league table" (communication with vendors).

LFRic infrastructure: Hierarchy of objects



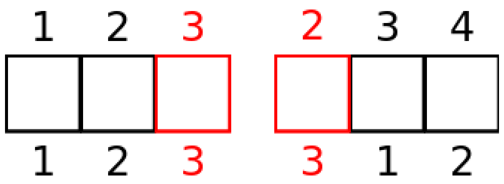
Met Office

LFRic Software Engineering: LFRic infrastructure

- **Support science** operations and **parallelism**:
  - Construct and handle **data objects** (e.g. mesh, reference element, function space, field);
  - **Support** for **distributed** and **shared memory** parallelism (e.g. dofmap, colouring for function spaces with shared DoFs, halo exchange).
- **Interface external libraries** (many more than in the UM – **collaboration** with other institutions + **in-house development** and implementation time)
  - **PSyclone** (STFC) for parallel code (& dependencies, e.g. Python);
  - **YAXT** (DKRZ) for MPI communications (& dependencies, e.g. MPICH);
  - **XIOS** (IPSL) for parallel IO (& dependencies, e.g. NetCDF, HDF5);
  - **pFUnit** (NASA Goddard) for unit testing;
  - **Rose picker** & **GPL-utilities** for Rose metadata; **Stylist** for style checking (in-house development based on external tools).

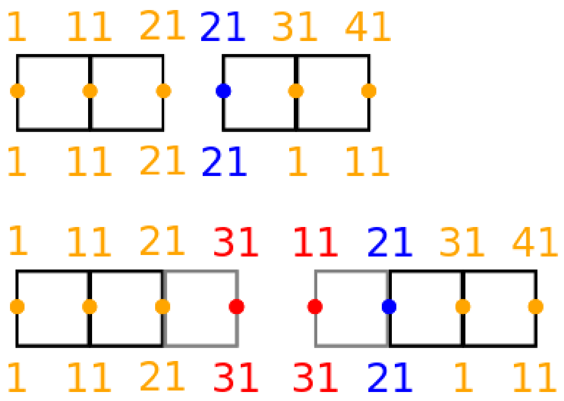
Met Office

Updating fields in parallel: continuous vs discontinuous spaces



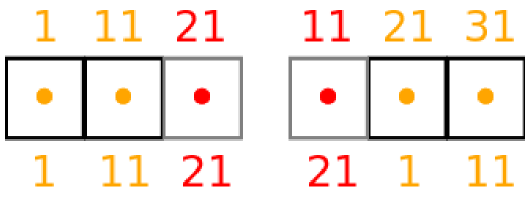
Continuous fields: PSy-layer cell loop, kernel calls

- DoFs on **owned** cells + redundant computation in the level-1 **halo**
- GH_INC and GH_READINC (R + W) access in kernel code – requires colouring for OpenMP



Continuous fields: PSy-layer DoF loop, built-in calls

- **Owned** DoFs or redundant computation into **annexed** DoFs (configuration option)



Discontinuous fields

- *Cell loop:* DoFs on **owned** cells (redundant computation optional)
- *DoF loop:* **owned** DoFs
- GH_READWRITE (R + W) or GH_WRITE (W) access – no colouring required for OpenMP

Examples of generated code

Code generation example

```
call invoke(                                              &
     setval_c(fld_c1, 0.0_r_def),                         &
     testkern_field_w2_inc_type(fld_c1, fld_c2), &
     testkern_field_w3_readwrite_type(fld_d1, fld_d2)  )
```

Algorithm code: invoke to kernels that update fields on a **continuous (W2)** and a **discontinuous (W3)** function space

```
 type(arg_type), dimension(2) :: meta_args = (/  &
      arg_type(gh_field, gh_real, gh_inc,   w2), &
      arg_type(gh_field, gh_real, gh_read, w1) )
integer :: iterates_over = cell_column
```

Kernel *testkern_field_w2_inc_mod* metadata: **continuous writer (W2)** and a continuous reader (W1)

```
 type(arg_type), dimension(2) :: meta_args = (/        &
     arg_type(gh_field, gh_real, gh_readwrite, w3), &
     arg_type(gh_field, gh_real, gh_read,         wtheta)  )
integer :: iterates_over = cell_column
```

Kernel *testkern_field_w2_inc_mod* metadata: **continuous writer (W3)** and a discontinuous reader (Wtheta)

PSyclone schedule: no distributed memory or optimisations

```
InvokeSchedule[invoke='invoke_0', dm=False]
    0: Loop[type='dof', field_space='any_space_1', it_space='dof', upper_bound='ndofs']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: BuiltIn setval_c(fld_c1,0.0_r_def)
    1: Loop[type='', field_space='w2', it_space='cell_column', upper_bound='ncells']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodedKern testkern_field_w2_inc_code(fld_c1,fld_c2) [module_inline=False]
    2: Loop[type='', field_space='w3', it_space='cell_column', upper_bound='ncells']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodedKern testkern_field_w3_readwrite_code(fld_d1,fld_d2) [module_inline=False]
```

Built-In call: loop over DoFs

Kernel calls: loops over cell columns

Met Office

Generated PSy layer: no distributed memory or optimisations

```
! Set-up all of the loop bounds
!
loop0_start = 1
loop0_stop = undf_aspc1_fld_c1
loop1_start = 1
loop1_stop = fld_c1_proxy%vspace%get_ncell()
loop2_start = 1
loop2_stop = fld_d1_proxy%vspace%get_ncell()
!
! Call our kernels
!
DO df=loop0_start,loop0_stop
  fld_c1_proxy%data(df) = 0.0_r_def
END DO
DO cell=loop1_start,loop1_stop
  !
  CALL testkern_field_w2_inc_code(nlayers, fld_c1_proxy%data, fld_c2_proxy%data, ndf_w2, undf_w2, map_w2(:,cell), ndf_w1, undf_w1, map_w1(:,cell))
END DO
DO cell=loop2_start,loop2_stop
  !
  CALL testkern_field_w3_readwrite_code(nlayers, fld_d1_proxy%data, fld_d2_proxy%data, ndf_w3, undf_w3, map_w3(:,cell), ndf_wtheta, undf_wtheta, map_wtheta(:,cell))
END DO
```

Built-In call loop bounds: number of unique DoFs

Kernel call loop bounds: number of cells

Inlined Built-In code

Kernel calls

Met Office

PSyclone schedule: distributed memory

```
InvokeSchedule[invoke='invoke_0', dm=True]
    0: Loop[type='dof', field_space='any_space_1', it_space='dof', upper_bound='nannexed']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: BuiltIn setval_c(fld_c1,0.0_r_def)
    1: HaloExchange[field='fld_c2', type='region', depth=1, check_dirty=True]
    2: Loop[type='', field_space='w2', it_space='cell_column', upper_bound='cell_halo(1)']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodedKern testkern_field_w2_inc_code(fld_c1,fld_c2) [module_inline=False]
    3: Loop[type='', field_space='w3', it_space='cell_column', upper_bound='ncells']
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodedKern testkern_field_w3_readwrite_code(fld_d1,fld_d2) [module_inline=False]
```

Built-In call: loop over DoFs, upper bound to last annexed DoF (LFRic API setting)

Halo exchange to depth 1 on a continuous reader

Kernel call on a **continuous FS**: loop over cell columns, upper bound halo depth 1

Kernel call on a **discontinuous FS**: loop over cell columns, upper bound number of owned cells in a partition

Generated PSy layer: distributed memory (1)

```
! Set-up all of the loop bounds
!
loop0_start = 1
loop0_stop = fld_c1_proxy%vspace%get_last_dof_annexed()
loop1_start = 1
loop1_stop = mesh%get_last_halo_cell(1)
loop2_start = 1
loop2_stop = mesh%get_last_edge_cell()
!
! Call kernels and communication routines
!
DO df=loop0_start,loop0_stop
  fld_c1_proxy%data(df) = 0.0_r_def
END DO
!
! Set halos dirty/clean for fields modified in the above loop
!
CALL fld_c1_proxy%set_dirty()
```

Built-In call loop bounds: last annexed DoF (LFRic API setting)

Kernel call loop bound **(continuous FS)**: last cell in depth-1 halo

Kernel call loop bound **(discontinuous FS)**: last owned (edge) cell

Inlined Built-In code

Flag updated halos

Met Office

Generated PSy layer: distributed memory (2)

```
      IF (fld_c2_proxy%is_dirty(depth=1)) THEN
        CALL fld_c2_proxy%halo_exchange(depth=1)
      END IF
      !
      DO cell=loop1_start,loop1_stop
        !
        CALL testkern_field_w2_inc_code(nlayers, fld_c1_proxy%data, fld_c2_proxy%data, ndf_w2, undf_w2, map_w2(:,cell), ndf_w1,
undf_w1, map_w1(:,cell))
      END DO
      !
      ! Set halos dirty/clean for fields modified in the above loop
      !
      CALL fld_c1_proxy%set_dirty()
      !
      DO cell=loop2_start,loop2_stop
        !
        CALL testkern_field_w3_readwrite_code(nlayers, fld_d1_proxy%data, fld_d2_proxy%data, ndf_w3, undf_w3, map_w3(:,cell),
ndf_wtheta, undf_wtheta, map_wtheta(:,cell))
      END DO
      !
      ! Set halos dirty/clean for fields modified in the above loop
      !
      CALL fld_d1_proxy%set_dirty()
```

Halo exchange to depth 1 on a continuous reader

Kernel call on a **continuous FS**

Flag updated halos

Kernel call on a **discontinuous FS**

Flag updated halos

Met Office

PSyclone schedule (part): distributed + shared memory

```
1: HaloExchange[field='fld_c2', type='region', depth=1, check_dirty=True]
  2: Loop[type='colours', field_space='w2', it_space='cell_column', upper_bound='ncolours']
    Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
    Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
    Schedule[]
      0: OMPParallelDirective[]
        Schedule[]
          0: OMPDoDirective[]
            Schedule[]
              0: Loop[type='colour', field_space='w2', it_space='cell_column', upper_bound='colour_halo(1)']
                Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                Schedule[]
                  0: CodedKern testkern_field_w2_inc_code(fld_c1,fld_c2) [module_inline=False]
  3: OMPParallelDirective[]
    Schedule[]
      0: OMPDoDirective[]
        Schedule[]
          0: Loop[type='', field_space='w3', it_space='cell_column', upper_bound='ncells']
            Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
            Literal[value:'NOT_INITIALISED', Scalar<INTEGER, UNDEFINED>]
            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
            Schedule[]
              0: CodedKern testkern_field_w3_readwrite_code(fld_d1,fld_d2) [module_inline=False]
```

Halo exchange to depth 1 on a continuous reader

Kernel call on a **continuous FS**: **outer loop** over colours, upper bound number of colours

OpenMP directives

Kernel call on a **continuous FS**: **inner loop** over cells in a colour, upper bound depth 1 halo for a colour

OpenMP directives

Kernel call on a **discontinuous FS**: loop over cell columns, upper bound number of owned cells in a partition

Generated PSy layer: distributed + shared memory (1)

```
ncolour = mesh%get_ncolours()
cmap => mesh%get_colour_map()
 …
loop0_start = 1
loop0_stop = fld_c1_proxy%vspace%get_last_dof_halo(1)
loop1_start = 1
loop1_stop = ncolour
loop2_start = 1
loop3_start = 1
loop3_stop = mesh%get_last_edge_cell()
!
! Call kernels and communication routines
!
!$omp parallel default(shared), private(df)
!$omp do schedule(static)
DO df=loop0_start,loop0_stop
  fld_c1_proxy%data(df) = 0.0_r_def
END DO
!$omp end do
!
! Set halos dirty/clean for fields modified in the above loop
!$omp master
CALL fld_c1_proxy%set_dirty()
CALL fld_c1_proxy%set_clean(1)
!$omp end master
!$omp end parallel
```

Get infrastructure colouring support

Built-In call loop bounds

Kernel call outer loop bound **(continuous FS)**: number of colours

Kernel call inner loop bound **(continuous FS)**: number of cells in a colour

Kernel call loop bound **(discontinuous FS)**: last owned (edge) cell

OpenMP directives

Inlined Built-In code

Flag updated halos

Generated PSy layer: distributed + shared memory (2)

```
IF (fld_c2_proxy%is_dirty(depth=1)) THEN
  CALL fld_c2_proxy%halo_exchange(depth=1)
END IF
!
DO colour=loop1_start,loop1_stop
  !$omp parallel default(shared), private(cell)
  !$omp do schedule(static)
  DO cell=loop2_start,last_cell_all_colours(colour,1)
    !
    CALL testkern_field_w2_inc_code(nlayers, fld_c1_proxy%data, fld_c2_proxy%data, ndf_w2,
undf_w2, map_w2(:,cmap(colour, cell)), ndf_w1, undf_w1, map_w1(:,cmap(colour, cell)))
  END DO
  !$omp end do
  !$omp end parallel
END DO
!
! Set halos dirty/clean for fields modified in the above loop
!
CALL fld_c1_proxy%set_dirty()
```

Halo exchange to depth 1 on a continuous reader

Kernel call **(continuous FS)**, outer loop over colours

OpenMP directives

Kernel call **(continuous FS)**, inner loop over cells in a colour

Kernel call on a **continuous FS**

Flag updated halos

Met Office

Generated PSy layer: distributed + shared memory (3)

```
!$omp parallel default(shared), private(cell)
!$omp do schedule(static)
DO cell=loop3_start,loop3_stop
  !
  CALL testkern_field_w3_readwrite_code(nlayers, fld_d1_proxy%data, fld_d2_proxy%data, ndf_w3,
undf_w3, map_w3(:,cell), ndf_wtheta, undf_wtheta, map_wtheta(:,cell))
END DO
!$omp end do
!
! Set halos dirty/clean for fields modified in the above loop
!
!$omp master
CALL fld_d1_proxy%set_dirty()
!$omp end master
!
!$omp end parallel
```

OpenMP directives

Kernel call **(discontinuous FS)**, loop over owned cells only

Flag updated halos