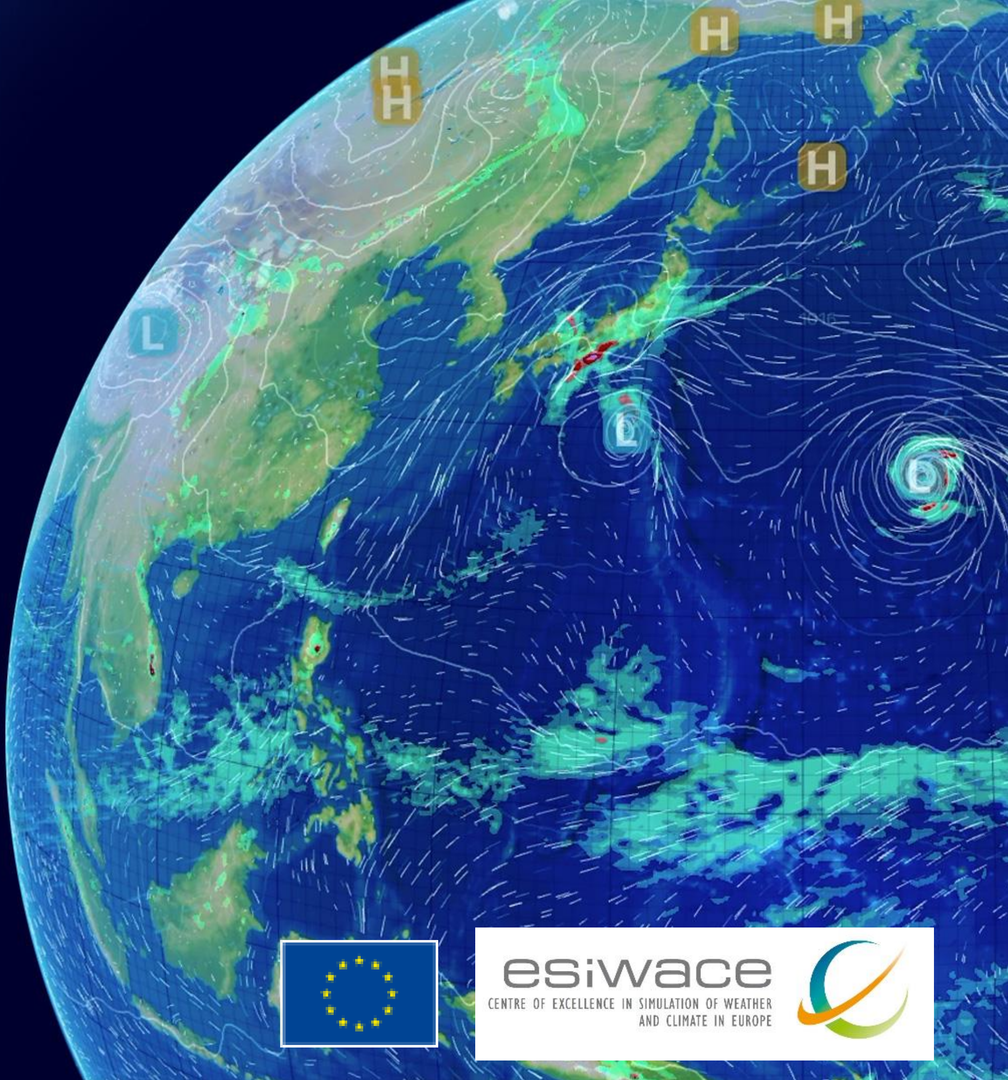
LFRic: Introduction and hands-on

Iva Kavcic, Met Office, UK &

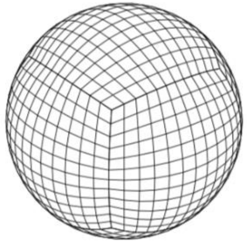
Rupert Ford, Andrew Porter, Sergi Siso (STFC, UK); Joerg Henrichs (BOM, AU); LFRic and DR Teams (Met Office, UK)

ESiWACE-2 DSL Training, 23 November 2020



LFRic (*after **L**ewis **F**ry **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

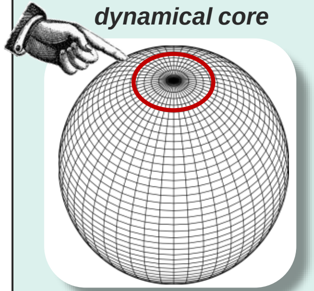
- Uses the **GungHo dynamical core**
- Runs on a **semi-structured cubed-sphere mesh**
- Uses **PSyclone** to generate **parallel code**



PSyclone

Increased resolution in NWP → Exascale computation

Unified Model & ENDGame dynamical core



Need a more scalable dynamical core

Need to make porting the codes from machine to machine easier: Flexible Implementation

New Dynamical Core – GungHo 工合

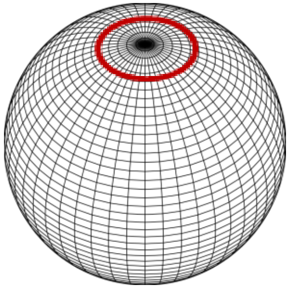
New Infrastructure – LFRic

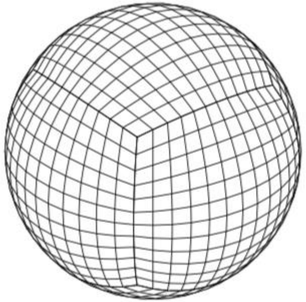
- **GungHo project:** Met Office, NERC (UK Universities) and STFC collaboration (2010 – 2015)
- **Future architectures?** – MPI, OpenMP Accelerators, GPUs, ARM, …?
- Complex parallel code + Complex parallel architectures + Complex compilers = **Complex optimisation space** → no single solution
- *Single source science code & performance portability(???) →* ***generated parallel code***

Conclusions from GungHo

- Keep the best of current MO ENDGame **dynamical core** and **improve** where possible (e.g. conservation)
  - Staniforth & Thuburn (2012) "Ten essential and desirable properties of a dynamical core"
  - Inspired by UM iterative-semi-implicit semi-Lagrangian scheme
- **Layered, "single-model" structure:** separate development teams for science (**GHASP**), infrastructure (**LFRic**) and parallelisation and optimisation (**PSyclone**)
- Optimisations provided by a code generator (***PSy + clone*** = *PSy layer generation*)
- Language: Object-orientated **Fortran 2003**
- Bring in **Physics parameterisations**: Reuse of UM code where possible; Couple these finite-different codes to the new finite-element core

Preparation for exascale: From UM/ENDGame to LFRic/GungHo





Unified Model (UM) & ENDGame dynamical core

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

LFRic system & GungHo dynamical core

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

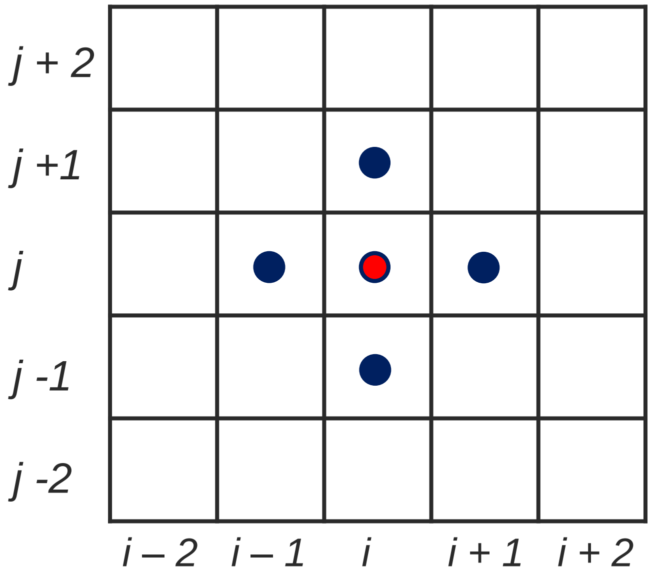
FDM on a structured mesh

- $f(x) \approx p_n(x), \frac{\mathrm{d}f(x)}{dx} \approx \frac{p'_n(x)}{\mathrm{dx}} + O(dx^{(n)})$
- Example: 2D Laplacian, equidistant mesh

$\nabla f(x_i, y_j) \approx [f(x_{i-1}, y_j) + f(x_i, y_{j-1})$ - $4f(x_i, y_j)$
$+ f(x_{i+1}, y_j) + f(x_i, y_{j+1})]/h^2$

→ Mesh layout explicit in the **stencil**: **location** and **connectivity**

→ Stencil representation relatively simple: *go_stencil(010,111,010)* in PSyclone GOcean 1.0 API
(depth and direction information for halo exchages)



$f(x_i, y_j)$

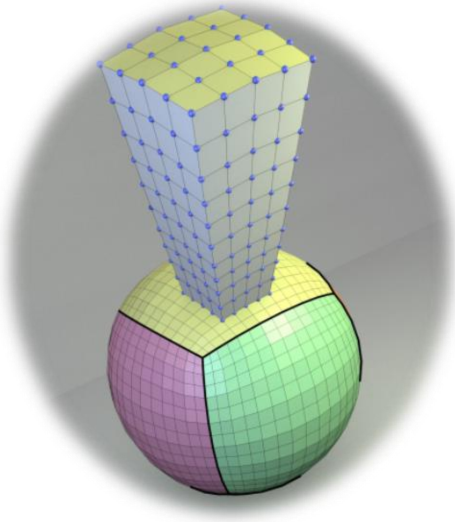
Mixed FEM on a **semi-structured mesh** (software engineers)

Global 2D Mesh
- Entities (vertices, edges, cells)
- Global IDs
- Connectivity (lookup arrays)

Partition
- Strategy (mesh type, ranks)
- Range of cells (owned and halos)
- Interface to distributed memory communication

Local 3D mesh
- Entities (vertices, edges, faces, cells)
- Local IDs
- Connectivity (lookup arrays)

2D cubed-sphere mesh extruded into 3D levels (*courtesy of Ricky Wong, animation on MO LFRic Wiki*)



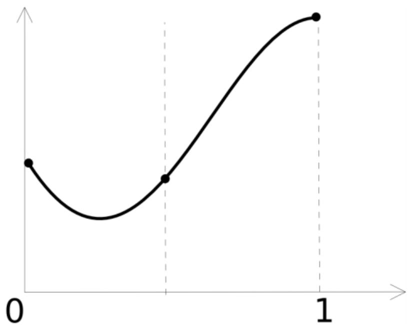
- Horizontal adjacency lost
- **Vertically** adjacent cells **contiguous** in memory → operate on **columns** of data

Mixed **FEM** on a semi-structured mesh

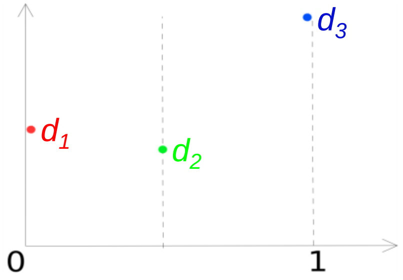
$$f(x) = \sum_{i=1}^{n} d_i \sigma_i(x)$$

Field

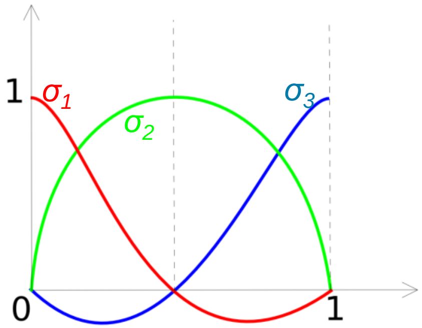
$$\sum_{i=1}^{3} =$$



$$\frac{df(x)}{dx} \sim \sum_{i=1}^{n} d_i \frac{d\sigma_i(x)}{dx}$$



×



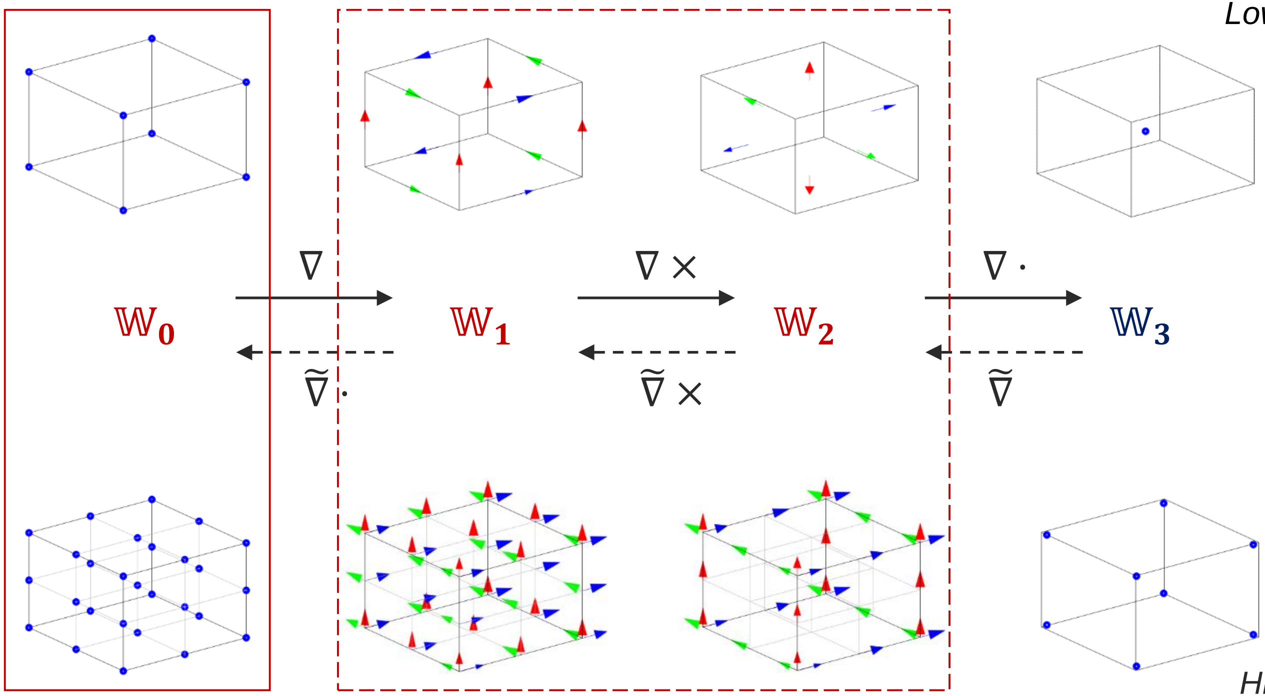
Degrees of Freedom ("dof"s)

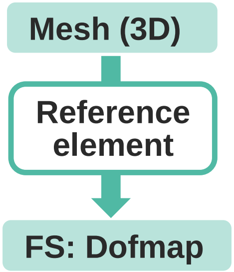
Function Space (FS)

Basis Functions

Mixed FEM (scientists): continuous + discontinuous

Lowest order (k = 0) reference element



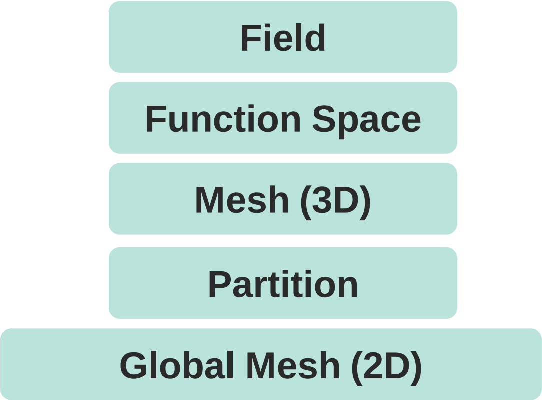


No shared dofs: W_3

Shared dofs: W_0 (fully), W_1 (tangential components) & W_2 (normal components)

Higher order (k = 1) reference element

LFRic hierarchy of classes (data + procedures)



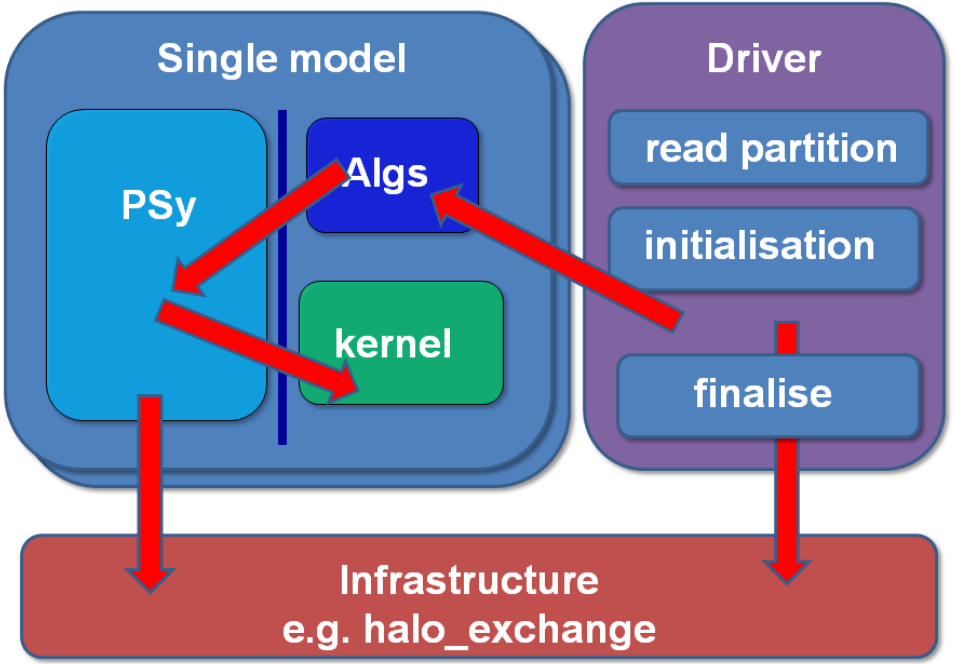
LFRic infrastructure: Hierarchy of classes

Role of infrastructure

- Provide **support** for **science** operations
- Construct and handle **data objects** (e.g. mesh, field, reference element, function space)
- **Support** for **distributed** and **shared memory** parallelism (e.g. dofmap, colouring for FS with shared dofs, halo exchange)
- **Interface** **external libraries** (e.g. YAXT for MPI communications, XIOS for parallel IO)

PSyKAl Separation of Concerns

between parallel code (*PSy*) and science code (*KAl*)



Algorithms (Natural Science: operations on ***whole field objects***)

⇩

Parallel-Systems (Computational Science: accesses field data and applies optimisations – ***generated code***)

⇩

Kernels (Natural Science: operations on ***data points***)

Use of “PSy” + clone in LFRic

- **Optimisations:** generates optimised PSy-layer code *(and soon transformed kernels – work in progress).*
- **Development**
  - Generates **kernel stubs** (argument declarations and ordering).
  - **fparser2** (F2003-2008): base for the **LFRic code style checker**.
- **Tools (profiling, DataAPI)**
  - Insert calls to profiling tools (interface in PSyclone) – tested (and used) in LFRic.
  - Extract data for running smaller code units as stand-alone applications (microbenchmarks) – work in progress.

Rules of engagement: LFRic ⟷ PSyclone Dynamo0.3 API

LFRic

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

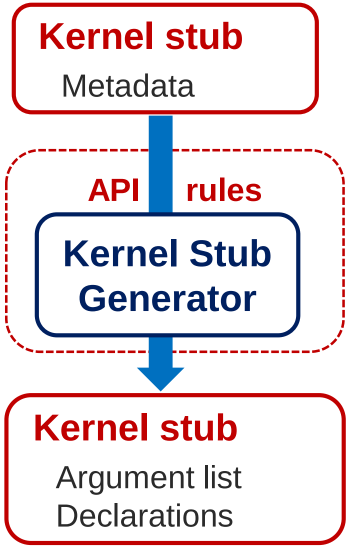
Dynamo0.3 API

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

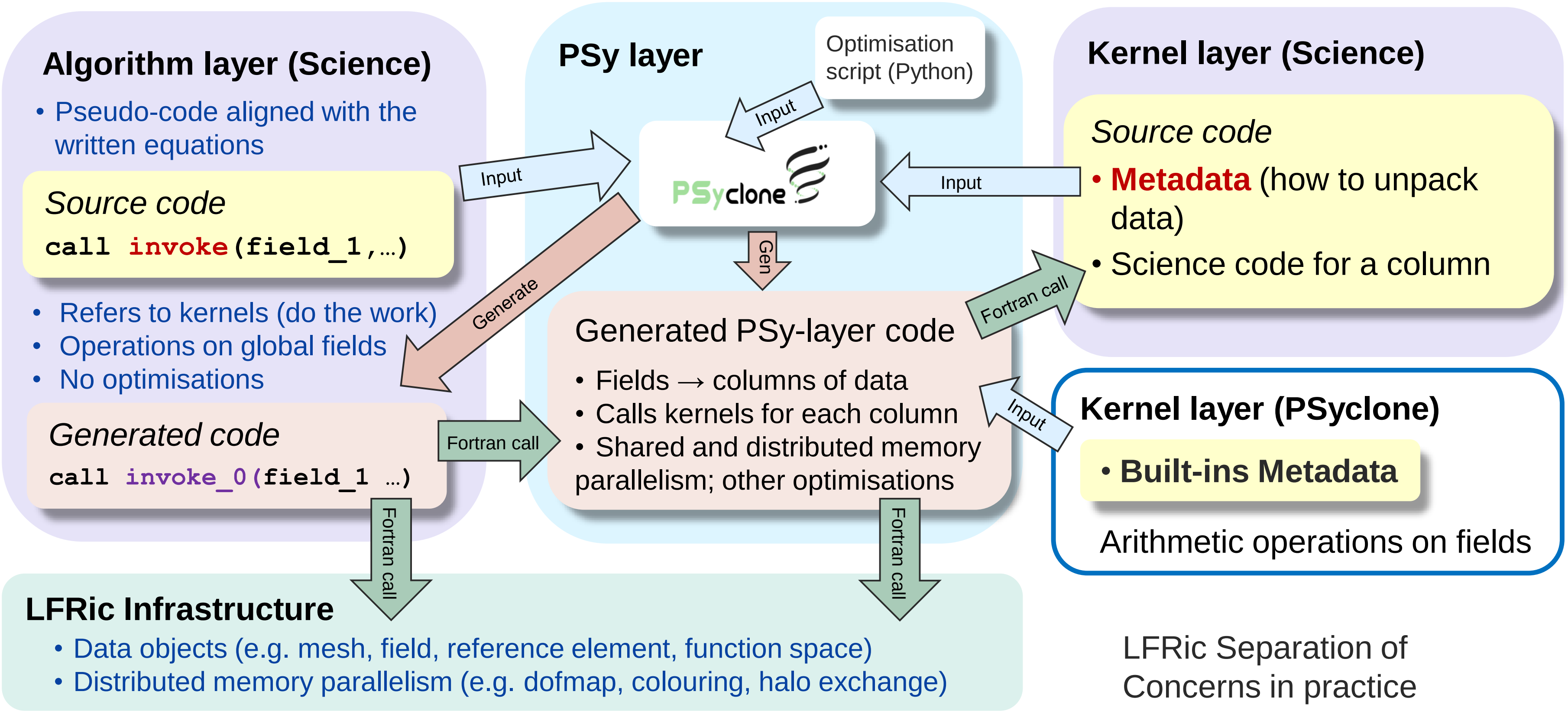
PSyKAl-lite

Required functionality (hand-written)

Kernel Stub Generator (R. Ford & A. Porter): *Metadata + Rules = Help in LFRic development*



DSL in action: LFRic – PSyclone communication



LFRic Separation of Concerns in practice

Met Office

LFRic: Algorithm code

Written by Scientists (DSL embedded in Fortran)

```
module lhs_alg_mod
…
 subroutine on_the_fly_lhs_alg(lhs, state, ref_state, … )
   use rtheta_kernel_mod,        only: rtheta_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) :: lhs(bundle_size)
   …
   call invoke( name = "Compute lhs_theta",                         &
                rtheta_kernel_type( lhs_tmp(igh_t), theta_ref, u, qr ), &
                matrix_vector_kernel_type( lhs(igh_t), theta, mm_wtheta ), &
                inc_X_plus_bY( lhs(igh_t), tau_t_dt, lhs_tmp(igh_t) )
  …
  end subroutine on_the_fly_lhs_alg
end module lhs_alg_mod
```

Global fields:

data layout hidden

Kernel (LFRic)

- Loops over columns of cells

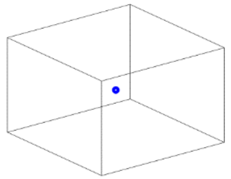
Built-in (PSyclone)

- Loops over all field dofs (arithmetic operations)

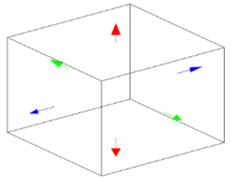
LFRic Kernel code *(Written by Scientists (Fortran))*

Metadata tells PSyclone how to unpack data

```
module rtheta_kernel_mod
…
  type, public, extends(kernel_type) :: rtheta_kernel_type
    private
    type(arg_type) :: meta_args(4) = (/                          &
         arg_type(GH_FIELD, GH_READWRITE, Wtheta),                &
         arg_type(GH_FIELD, GH_READ,      ANY_DISCONTINUOUS_SPACE_1), &
         arg_type(GH_FIELD, GH_INC,       W2),                    &
         arg_type(GH_FIELD, GH_READ,      ANY_SPACE_1),           &
         /)
    type(func_type) :: meta_funcs(2) = (/                        &
         func_type(Wtheta, GH_BASIS, GH_DIFF_BASIS),              &
         func_type(W2,     GH_BASIS, GH_DIFF_BASIS)               &
         /)
    integer :: iterates_over = CELLS
    integer :: gh_shape = GH_QUADRATURE_XYoZ
  contains
    procedure, nopass ::rtheta_code
  end type
…
```

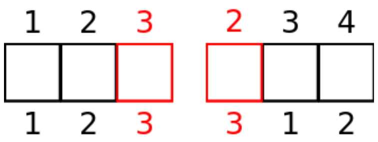


Discontinuous *function spaces: no shared DoFs*



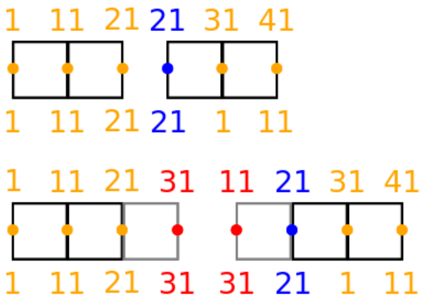
Continuous *function spaces: shared DoFs*

Updating fields in parallel in PSy layer: continuous vs discontinuous spaces



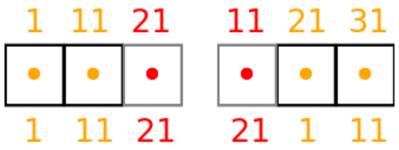
Continuous fields: *PSy-layer* ***cell loop****, kernel calls*

- Dofs on **owned** cells + redundant computation in the level-1 **halo**
- **GH_INC (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

LFRic: Kernel code

Written by Scientists ("Fortran 90" style – **no objects allowed!**)

```
…
  subroutine rtheta_code( … r_theta, heat_flux, u, &
…sizes, maps, basis functions,
 quadrature points for all function spaces )
…
    real(kind=r_def), dimension(undf_wtheta), &
                      intent(inout) :: r_theta
    real(kind=r_def), dimension(undf_2d),     &
                      intent(in)    :: heat_flux
    real(kind=r_def), dimension(undf_w2),     &
                      intent(in)    :: u
…
    do k = 0, nlayers-1
      do df = 1, ndf_wtheta
        r_theta( map_wtheta(df) + k ) =  &
        r_theta( map_wtheta(df) + k ) - rtheta_e(df)
      end do
    end do
…
  end subroutine rtheta_code
end module rtheta_kernel_mod
```

Science code for a **column of *nlayers* levels**:

- **k-outer** over cells (layers)
- **dof-inner** over DoFs in a cell

Work in progress:

- *Kernel transformations & loop re-ordering (PSyclone)*
- *i-first fields for Physics parameterisation schemes (LFRic infrastructure and PSyclone)*

Hands-on sessions

https://github.com/stfc/PSyclone/blob/master/tutorial/practicals/LFRic/building_code/README.md

LFRic layers and what we are working on

1. Creating and calling kernels

2. Using built-ins

3. Time evolution of a field (time-permitting)

LFRic Driver layer overview – not for hands-on tutorial

Driver layer: set up and control of a model run

- Set up of the LFRic object stack: **global 2D mesh → partition → local 3D partitioned mesh → function space → field**
- Model initialisation (science configurations, initial data)
- Controls of a model run (e.g. time-step loop, checkpoint) includes wrappers to external libraries (e.g. YAXT, XIOS)

Simplified example in Tutorial 3

LFRic layers for hands-on tutorials

- **Algorithm layer**: Operations on whole field (and other) objects using ***invoke*** DSL syntax
- **Kernel layer**: Operations on field (and other) object data using ***metadata*** DSL syntax and ***loops*** to update data
- **PSy layer**: From algorithms to kernels
  - **Unpacks and access object data via accessor classes (*proxy*)**
  - **Calls kernels for each column**
  - Shared and distributed memory parallelism *(see sessions on distributed and shared memory support)*

Tutorial 1, Part 1: Create simple kernels to update fields on specific LFRic function spaces

https://github.com/stfc/PSyclone/tree/master/tutorial/practicals/LFRic/building_code/1_simple_kernels/part1

- Open the README.md document in a browser tab
- Open the LFRic kernel structure document in a browser tab
- Open a terminal, make sure that the hands-on environment is working and navigate to

  `tutorial/practicals/LFRic/building_code/1_simple_kernels/part1`
- We will follow steps in the README.md document

Tutorial 1, Part 2: Create simple kernels to update fields on generic LFRic function spaces

https://github.com/stfc/PSyclone/tree/master/tutorial/practicals/LFRic/building_code/1_simple_kernels/part2

- Open the README.md document in a browser tab
- Open the LFRic kernel structure document in a browser tab
- Open a terminal, make sure that the hands-on environment is working and navigate to

  `tutorial/practicals/LFRic/building_code/1_simple_kernels/part2`

- We will follow steps in the README.md document

Tutorial 2: Using built-ins

https://github.com/stfc/PSyclone/tree/master/tutorial/practicals/LFRic/building_code/2_built_ins

- Open the README.md document in a browser tab
- Open the PSyclone LFRic (Dynamo 0.3 API) built-ins documentation in a browser tab
- Open a terminal, make sure that the hands-on environment is working and navigate to

  `tutorial/practicals/LFRic/building_code/2_built_ins`
- We will follow steps in the README.md document

Met Office

Tutorial 3: Time evolution of a field on a planar mesh

https://github.com/stfc/PSyclone/tree/master/tutorial/practicals/LFRic/building_code/3_time_evolution

- Open the README.md document in a browser tab
- Open a terminal, make sure that the hands-on environment is working and navigate to

  `tutorial/practicals/LFRic/building_code/3_time_evolution`

- We will follow steps in the README.md document

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

Questions?

Acknowledgements

LFRic team, GungHo Atmospheric Science team and other LFRic developers

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

Links and references

- LFRic: https://code.metoffice.gov.uk/trac/lfric/wiki
- LFRic container recipes (to be hosted on https://github.com/MetOffice):
  - https://github.com/eth-cscs/ContainerHackathon/tree/master/LFRIC
  - https://github.com/tinyendian/lfric_reader
- PSyclone and fparser
  - https://github.com/stfc/PSyclone
  - https://psyclone.readthedocs.io
  - https://github.com/stfc/fparser
  - https://fparser.readthedocs.io
- PSyclone in LFRic: https://code.metoffice.gov.uk/trac/lfric/wiki/PSycloneTool
- GHASP (GungHo Atmospheric Science): https://code.metoffice.gov.uk/trac/lfric/wiki/GungHoScience
- 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