Science and Technology Facilities Council

PSyclone

PSyIR: The PSyclone Internal Representation

Rupert Ford, Andy Porter, **Sergi Siso**, STFC Hartree Centre
Iva Kavcic, Chris Maynard, Andrew Coughtrie, UK Met Office
Joerg Henrichs, Australian Bureau of Meteorology

ESIWACE2 training course on Domain-specific Languages in Weather and Climate, 23rd-27th November 2020

Hartree Centre
Science & Technology Facilities Council

Met Office

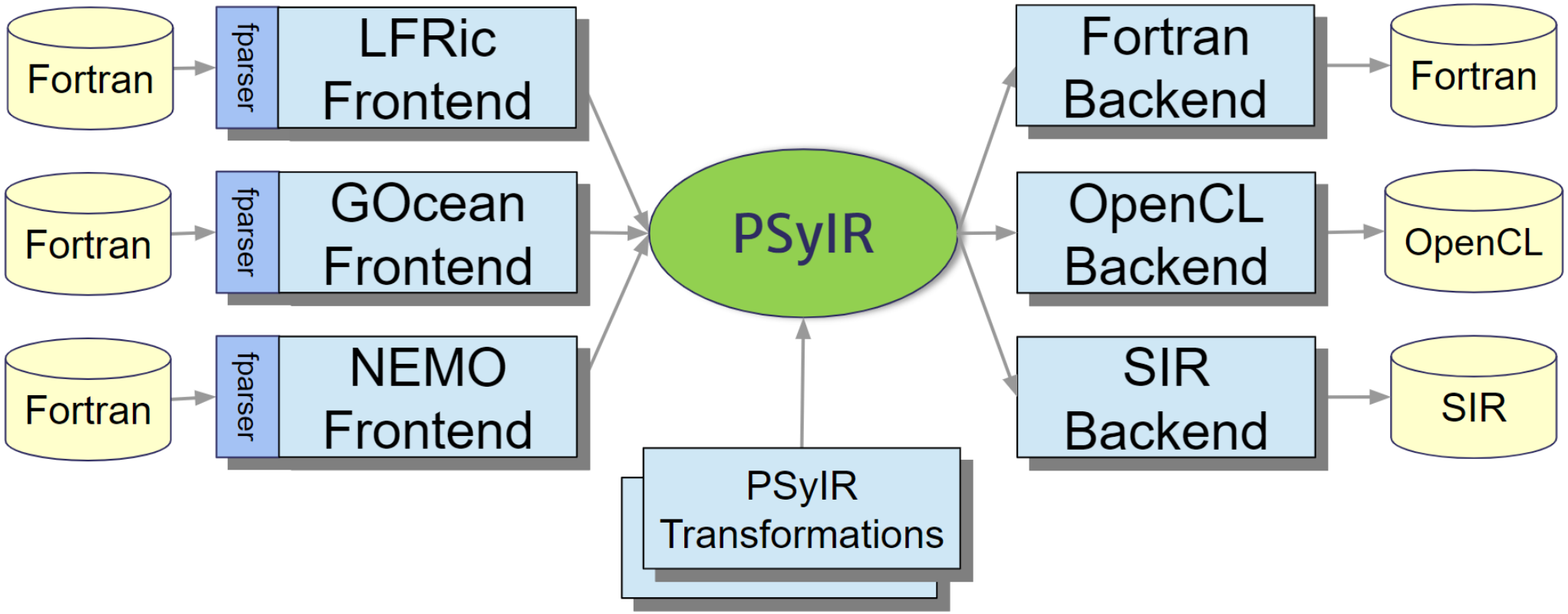
Australian Government
Bureau of Meteorology

Overview

1. What is the PSyIR
2. PSyIR structure and basic API
3. How PSyclone uses the PSyIR

1. What is the PSyIR

Vision

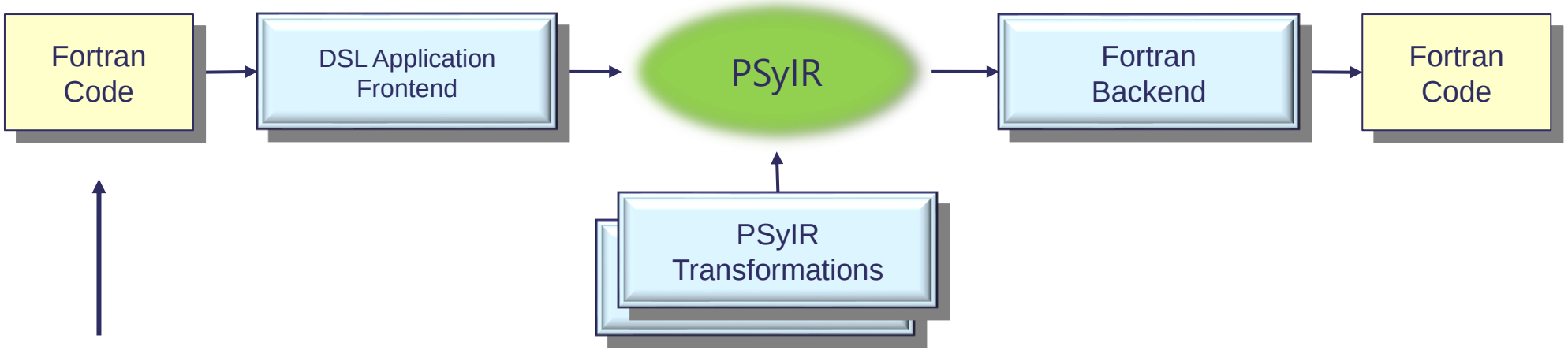


Hartree Centre – US Exascale Computing Project collaboration funded by STFC (UKRI)

Science and Technology Facilities Council

EUROEXA has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 754337
ESiWACE2 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823988

PSyclone workflow



Embedded DSL with the scientific domain knowledge

Parallelisation and optimisation encoded as ‘recipes’ (Python scripts).

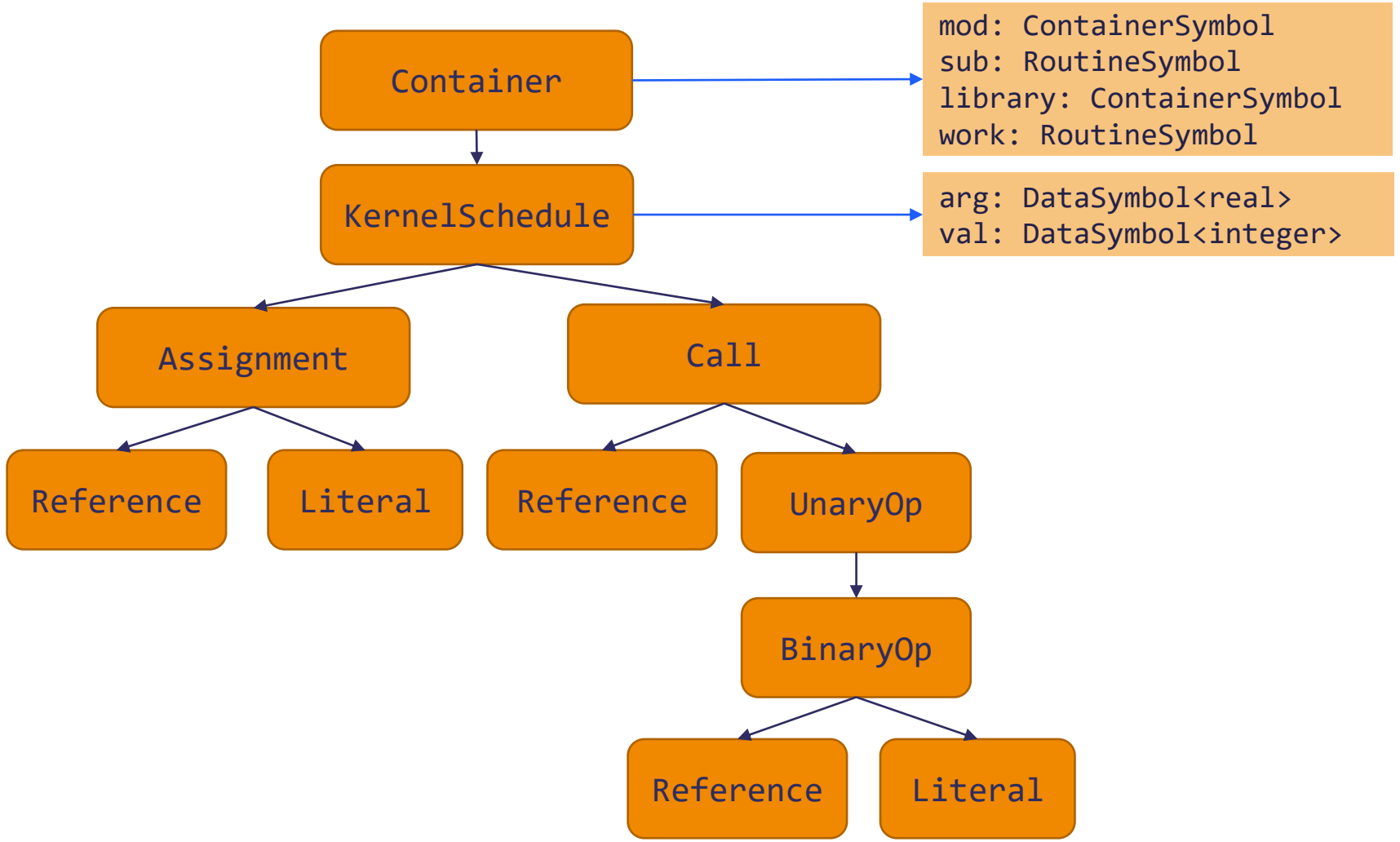
PSyIR

- It is a **language-independent** Intermediate Representation.
- It is a **mutable** representation intended to be programmatically manipulated through transformations or PSyclone scripts.
- It is itself **domain-agnostic**, but it is **extensible** to create the domain-specific DSLs that will be used by the applications.

2. PSyIR structure and basic API

PSyIR Structure:
Abstract Syntax Tree with Scoped Symbol Tables

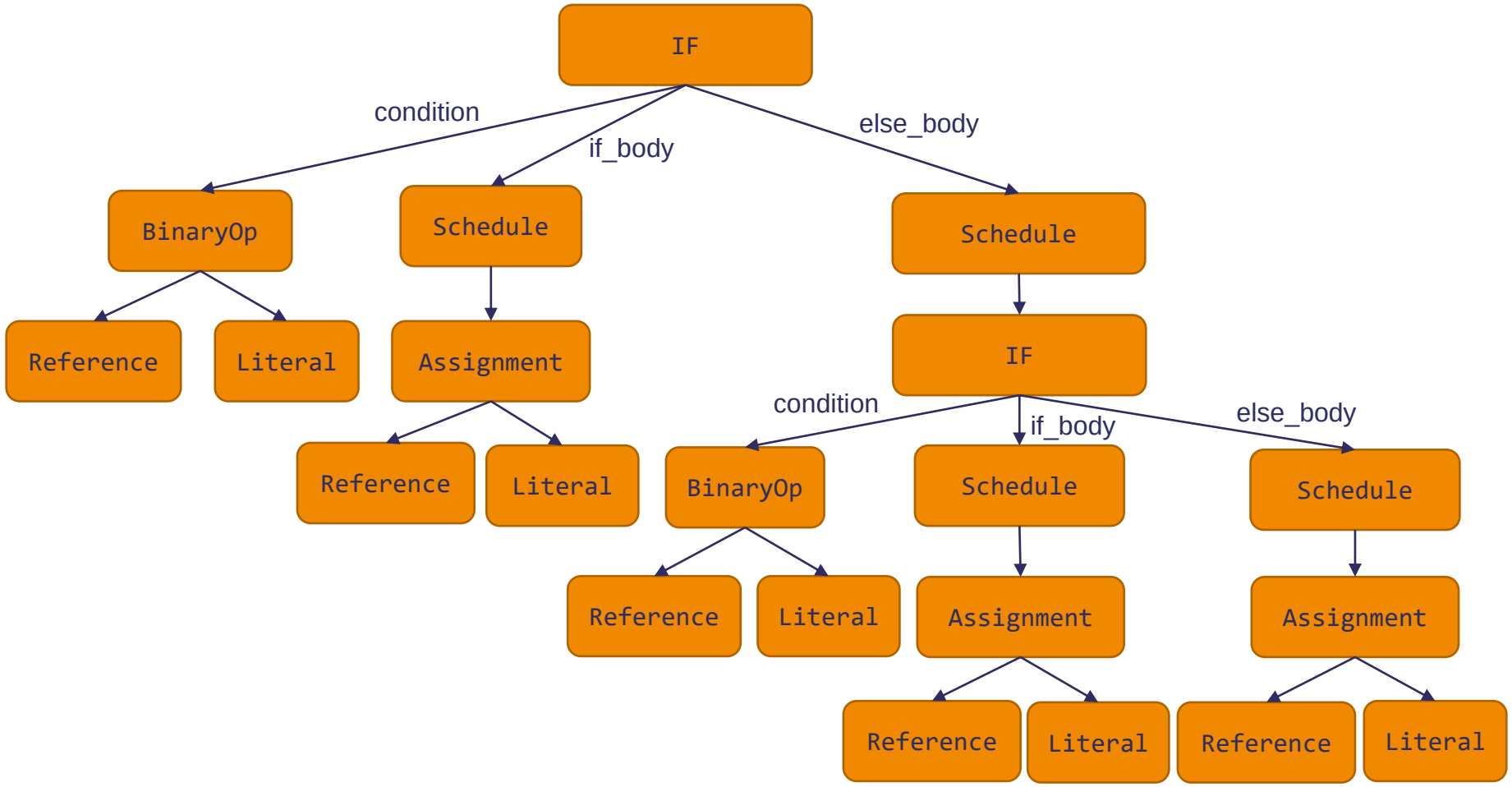
```
module mod
  use library, only : sub
contains
  subroutine work(arg)
    real, intent(inout) :: arg
    integer :: val

    val = 3
    call sub(val, SIN(arg + 2))
  end subroutine work
end module mod
```



Canonicalisation

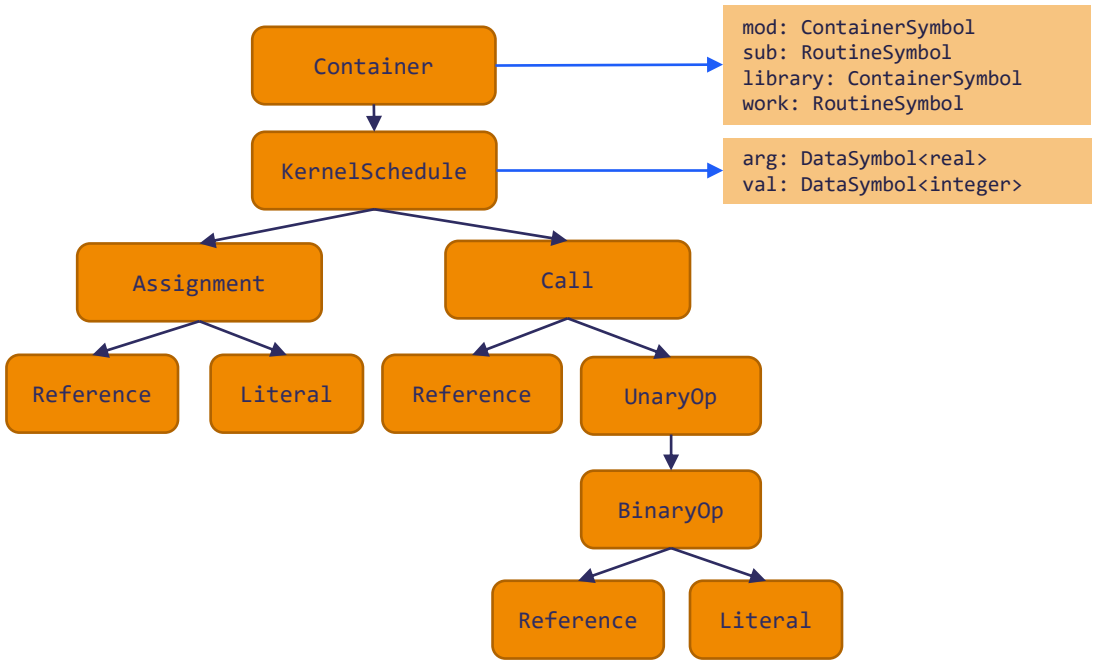
- The PSyIR only has 1 branching and 1 looping node. Syntactic constructs like: else if, do while, select/switch, where… are converted to these building blocks.

```
if (x < 0) then
  value = 1
else if ( x == 0) then
  value = 2
else
  value = 3
end if
```



Visualization (the *view()* method)

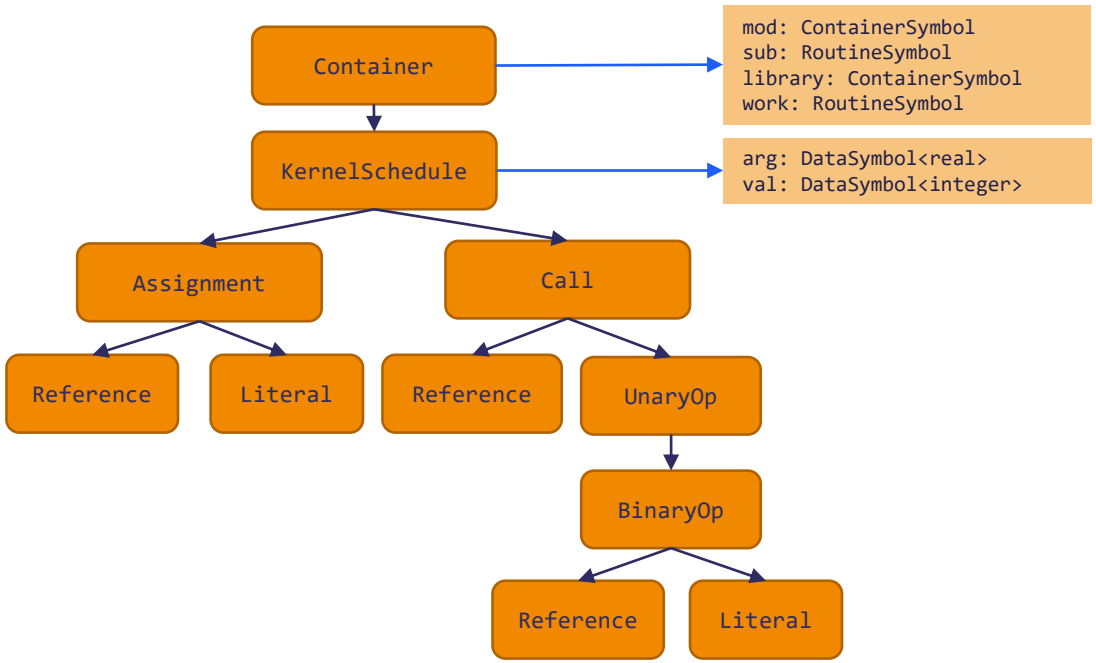
Install the *termcolor* optional dependency to display colourised output




```
(Pdb) CONTAINER.symbol_table.view()
Symbol Table:
library: <not linked>
sub : RoutineSymbol
mod
work : RoutineSymbol

(Pdb) KERNEL_SCHEDULE.symbol_table.view()
Symbol Table:
arg: <Scalar<REAL, UNDEFINED>, Argument(pass-by-value=False)>
val: <Scalar<INTEGER, UNDEFINED>, Local>

(Pdb) CONTAINER.view()
Container[mod]
    KernelSchedule[name:'work']
        0: Assignment[]
            Reference[name:'val']
            Literal[value:'3', Scalar<INTEGER, SINGLE>]
        1: Call[name='sub']
            Reference[name:'val']
            UnaryOperation[operator:'SIN']
                BinaryOperation[operator:'ADD']
                    Reference[name:'val']
                    Literal[value:'2.0', Scalar<REAL, UNDEFINED>]
```

Science and Technology Facilities Council

Navigation



- Homogeneous navigation:

.parent, .root, .children

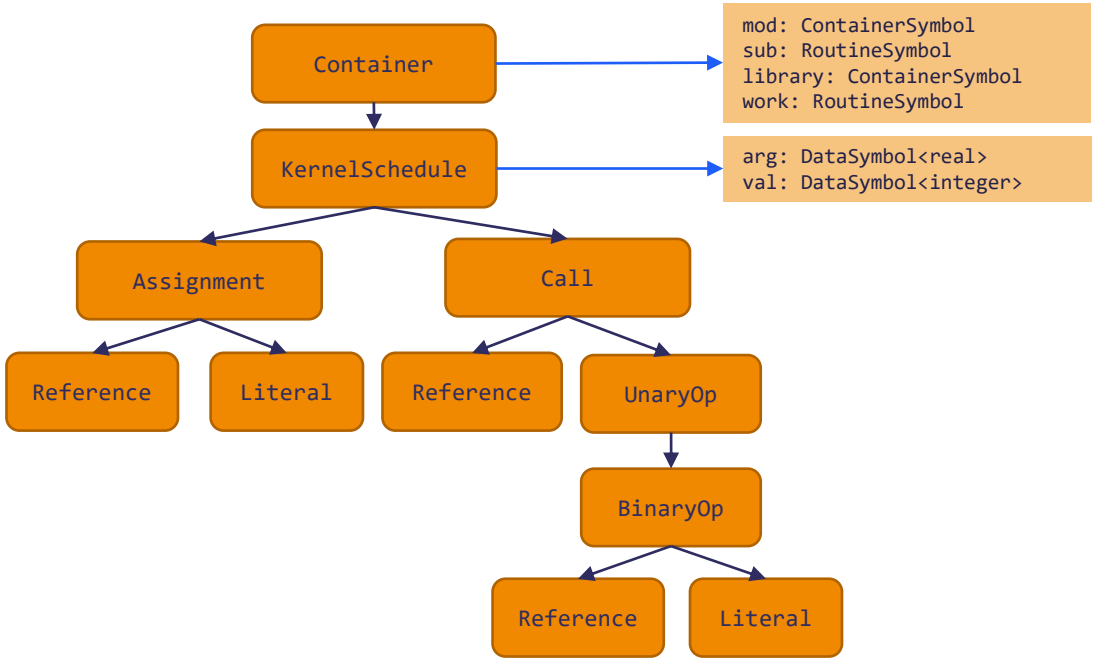
```
>>> BINARYOPERATION.parent
<psyclone.psyir.nodes.operation.UnaryOperation object at 0x7fe2ea561250>
>>> BINARYOPERATION.root
<psyclone.psyir.nodes.container.Container object at 0x7fe2ea5613d0>
>>> for child in BINARYOPERATION.children:
...     child
...
<psyclone.psyir.nodes.reference.Reference object at 0x7fe2ea561190>
<psyclone.psyir.nodes.literal.Literal object at 0x7fe2ea5c1fd0>
```

- Searching methods:

walk(type), ancestor(type): to search down- and up-wards

```
(Pdb) CONTAINER.walk(Literal)
[<psyclone.psyir.nodes.literal.Literal object at 0x7f5fd3eef3d0>, <psyclo
ne.psyir.nodes.literal.Literal object at 0x7f5fd3eef370>]
(Pdb) CONTAINER.walk(Literal)[1].ancestor(UnaryOperation)
<psyclone.psyir.nodes.operation.UnaryOperation object at 0x7f5fd3eef610>
```

Navigation 2



- Semantic navigation:

(semantic properties depending on node kind)

- Assignments

```
(Pdb) ASSIGN1.lhs
<psyclone.psyir.nodes.reference.Reference object at 0x7f27a1112550>
(Pdb) ASSIGN1.rhs
<psyclone.psyir.nodes.literal.Literal object at 0x7f27a11123d0>
```

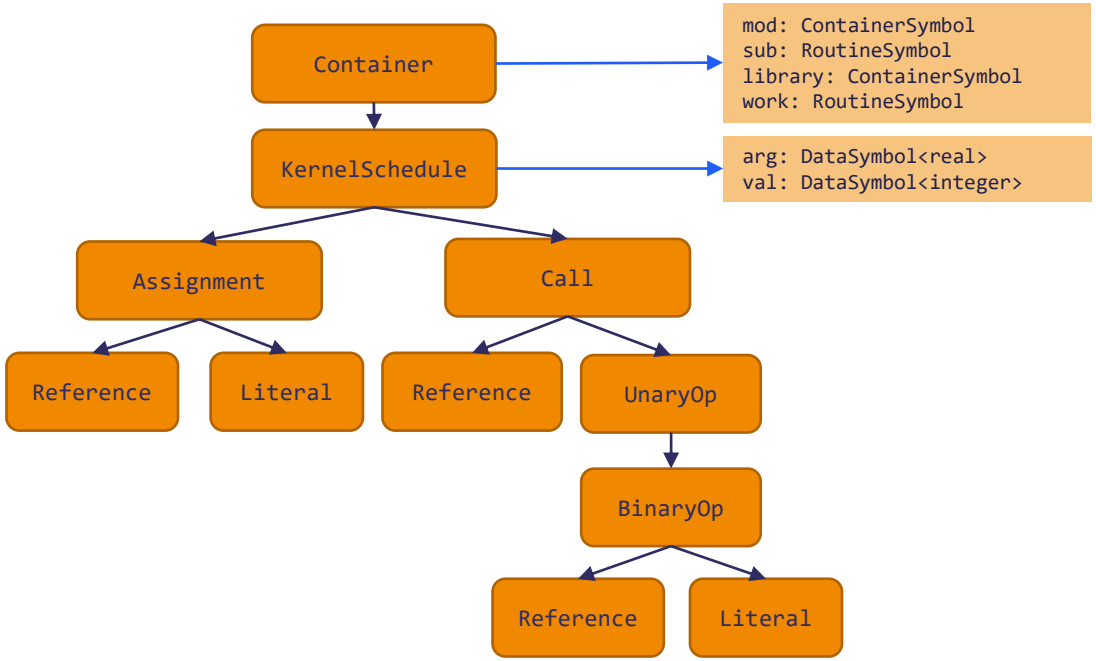
- Conditionals (not in given example tree)

```
(Pdb) IF.condition.view()
BinaryOperation[operator:'GT']
    Reference[name:'psyir_tmp']
    Literal[value:'0.0', Scalar<REAL, UNDEFINED>]
(Pdb) IF.if_body
<psyclone.psyir.nodes.schedule.Schedule object at 0x7ff240ba9610>
(Pdb) IF.else_body
```

- Loops (not in given example tree)

```
(Pdb) print(LOOP.start_expr)
Literal[value:'0', Scalar<INTEGER, SINGLE>]
(Pdb) print(LOOP.stop_expr)
Literal[value:'1', Scalar<INTEGER, 8>]
(Pdb) print(LOOP.step_expr)
Literal[value:'1', Scalar<INTEGER, 8>]
(Pdb) LOOP.loop_body
<psyclone.psyir.nodes.schedule.Schedule object at 0x7ff240ba9790>
```

Symbol Table



- All nodes can find their scope (and symbol table)

.scope

```
(Pdb) BINARYOPERATION.scope.symbol_table.view()
Symbol Table:
arg: <Scalar<REAL, UNDEFINED>, Argument(pass-by-value=False)>
val: <Scalar<INTEGER, UNDEFINED>, Local>
```

- Symbol tables recursively lookup symbols in parent symbol tables.

.lookup()

```
(Pdb) BINARYOPERATION.scope.symbol_table.lookup("library")
<psyclone.psyir.symbols.containersymbol.ContainerSymbol object at 0x7f6f1d849100>
```

- Some nodes have references to relevant symbols

e.g .routine, .symbol

```
(Pdb) CALL.routine
<psyclone.psyir.symbols.routinesymbol.RoutineSymbol object at 0x7f6f1d8491f0>
(Pdb) REF_VAL.symbol
<psyclone.psyir.symbols.datasymbol.DataSymbol object at 0x7f6f1d849400>
(Pdb)
```

Type System

```
(Pdb) BINARYOPERATION.scope.symbol_table.view()
Symbol Table:
arg: <Scalar<REAL, UNDEFINED>, Argument(pass-by-value=False)>
val: <Scalar<INTEGER, UNDEFINED>, Local>
```

- Import pre-defined Scalar Types:

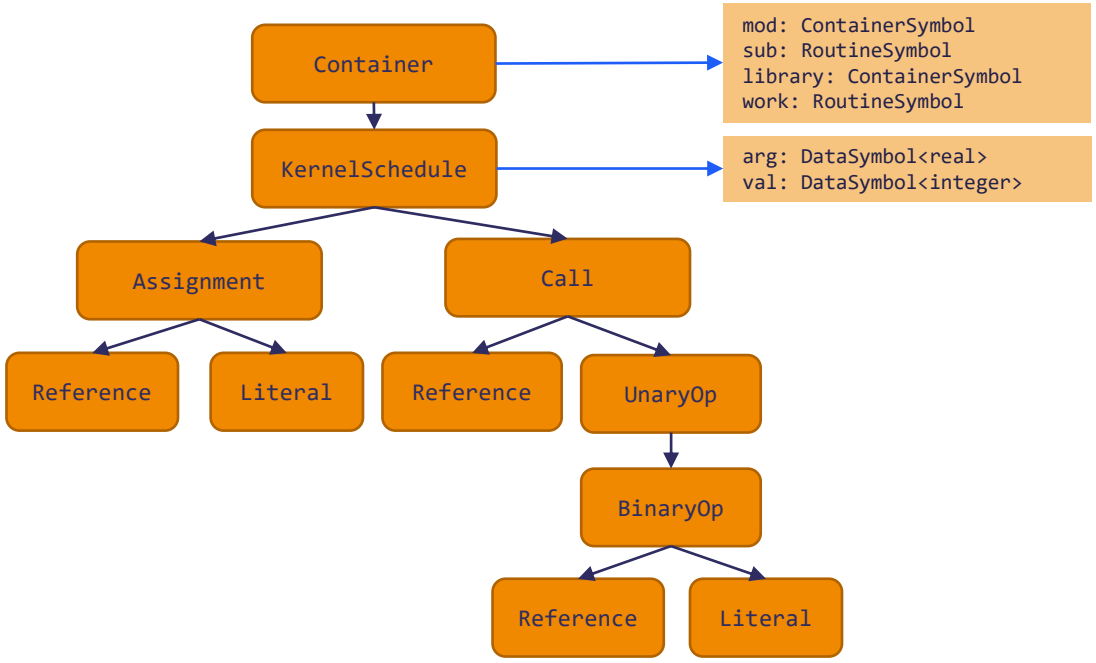
```
(Pdb) from psyclone.psyir.symbols import INTEGER_TYPE, REAL_SINGLE_TYPE, REAL_DOUBLE_TYPE, BOOLEAN_TYPE
```

- Create more complex types:

```
(Pdb) my_array_type = ArrayType(REAL_DOUBLE_TYPE, shape=[10, 10])
(Pdb) my_structure_type = StructureType()
(Pdb) my_structure_type.add("data", my_array_type, Symbol.Visibility.PUBLIC)
(Pdb) my_structure_type.add("length", INTEGER_TYPE, Symbol.Visibility.PUBLIC)
(Pdb) my_structure_type.add("flag_written", BOOLEAN_TYPE, Symbol.Visibility.PRIVATE)
```

Node Creation

Alternative to parse existing code



- Find accepted children

_children_*valid*_format

```
(Pdb) IfBlock._children_valid_format
'DataNode, Schedule [, Schedule]'
(Pdb) BinaryOperation._children_valid_format
'DataNode, DataNode'
```

- Use constructor for leaf nodes an top-down creation.
- Use .create() for Bottom-up creation.

```
(Pdb) lit1 = Literal("1", INTEGER_TYPE)
(Pdb) ref1 = Reference(TMP_SYMBOL)
(Pdb) asg = Assignment.create(ref1, lit1)
(Pdb) asg.view()
Assignment[]
    Reference[name:'psyir_tmp_1']
    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
```

Science and Technology Facilities Council

Note: If possible avoid low-level manipulation of the AST and use transformations (shown later)

CodeBlocks

- The PSyIR **CodeBlock** node contains unrecognised language-specific blocks of source code.

```
Schedule[]
    0: Loop[type='lon', field_space='None', it_space='None']
        Literal[value:'2', Scalar<INTEGER, UNDEFINED>]
        BinaryOperation[operator:'SUB']
            Reference[name:'jpi']
            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: CodeBlock[[<class 'fparser.two.Fortran2003.Write_Stmt'>]]
```

- Similarly, the UnknownType provides a solution for unrecognized type declarations.

Transformations

Avoid manual manipulations of the AST by using the available transformations

- List available transformations

```
(Pdb) from psyclone.psyGen import TransInfo
(Pdb) t = TransInfo()
```

Currently **TransInfo()** being refactored, check for transformation in the following locations:

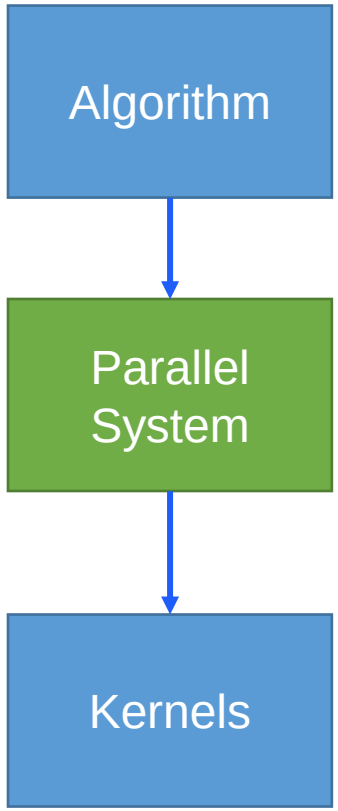
```
from psyclone.psyir.transformations import <transformation_name>
from psyclone.transformations import <transformation_name>
from psyclone.domain.<API_NAME>.transformations import transformation_name
```

- Va

```
(Pdb) omp_transformation = t.get_trans_name('OMPParallelLoopTrans')
(Pdb) omp_transformation.validate(LOOP)
(Pdb) omp_transformation.apply(LOOP)
```

3. How PSyclone uses the PSyIR

How PSyclone *currently* uses the PSyIR

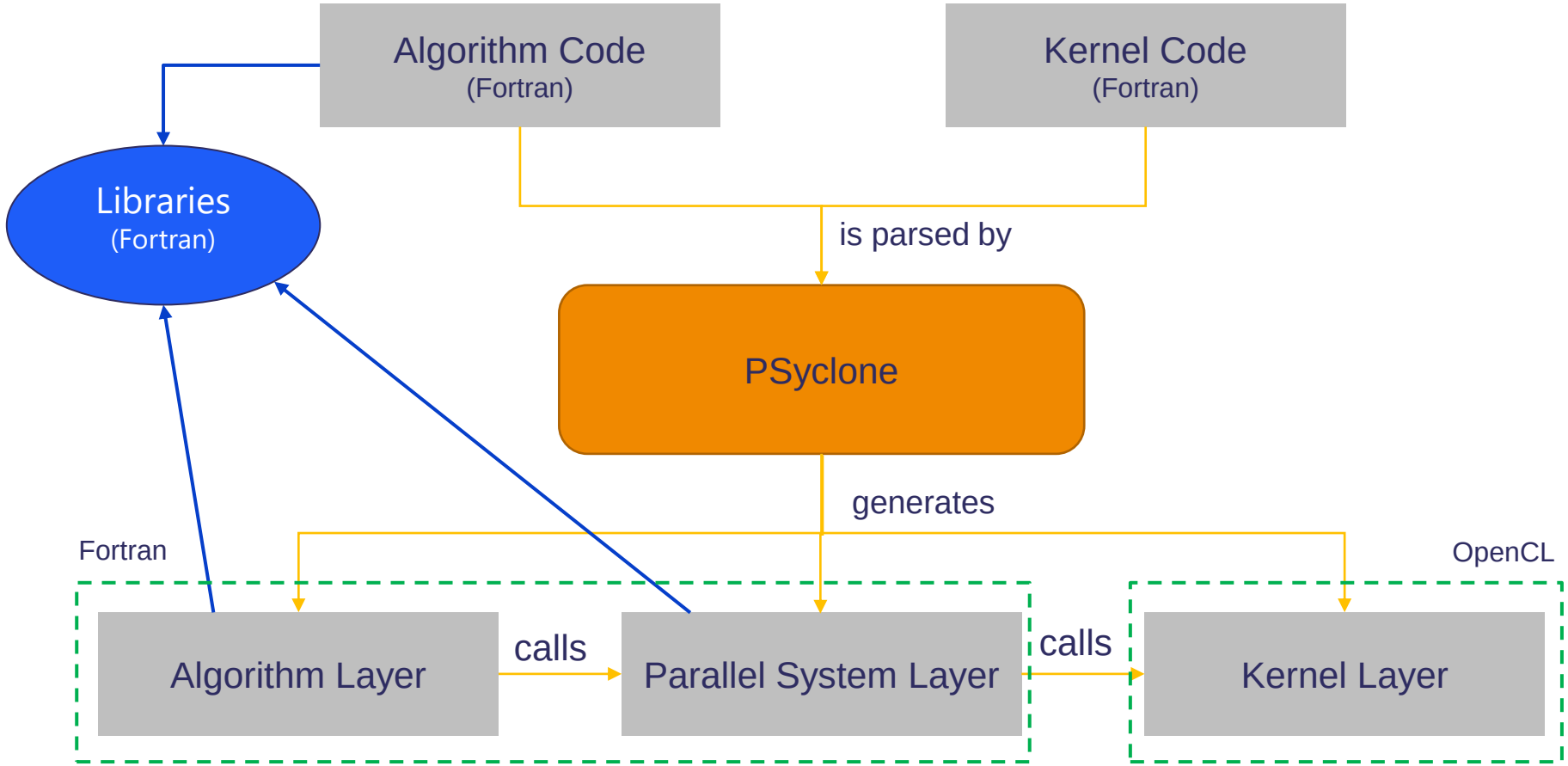


- No PSyIR Support yet.
- Existing code modified by PSyclone

- PSyIR with specialised domain-specific nodes and parallel constructs:
  - Directive, GlobalSum, HaloExchange, CodedKern, InlineKern, BuiltInKern, …
  - These classes are currently not compatible with the PSyIR backends but can generate the equivalent Fortran.
- Code fully-generated by PSyclone

- Full PSyIR support.
- Existing code modified by PSyclone

Fortran-to-OpenCL example



Note: More on OpenCL and other languages in the Wednesday talk “Expanding PSyclone target languages to leverage the wider HPC software ecosystem”

PSyIR in PSyclone Scripts

```
$ psyclone -s ./psyclone_script.py algorithm.f90
```

```
 1 def trans(psy):
 2     '''PSyclone script example to access Invoke and Kernels PSyIR.'''
 3
 4     # Loop over all of the Invokes in the PSy object.
 5     for invoke in psy.invokes.invoke_list:
 6         invoke_schedule = invoke.schedule
 7         invoke_schedule.view()
 8
 9         # Loop over all of the Kernels in this Invoke.
10         for kernelcall in invoke_schedule.coded_kernels():
11             kernel_schedule = kernelcall.get_kernel_schedule()
12             kernel_schedule.view()
13
14     return psy
```

PSy-layer PSyIR

Kernel PSyIR

Science and Technology Facilities Council

A complete example (part 1)

```
1   call invoke(                                                    &
2                continuity(ssha_t, sshn_t, sshn_u, sshn_v,         &
3                           hu, hv, un, vn),                        &
4                momentum_u(ua, un, vn, hu, hv, ht,                 &
5                           ssha_u, sshn_t, sshn_u, sshn_v),        &
```

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                Schedule[]
                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
    1: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
```

A complete example (part 2)

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
        Schedule[]
            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                Schedule[]
                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
    1: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
```

```
def trans(psy):
    ''' Transformation entry point '''
    tinfo = TransInfo()
    loop_trans = tinfo.get_trans_name('GOceanOMPLoopTrans')
    parallel_trans = tinfo.get_trans_name('OMPParallelTrans')

    schedule = psy.invokes.get('invoke_0').schedule

    for child in schedule.children:
        schedule, _ = loop_trans.apply(child)

    schedule, _ = parallel_trans.apply(schedule.children)

    return psy
```

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Directive[OMP parallel]
        Schedule[]
            0: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
                        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
                            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                                Schedule[]
                                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
            1: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
                        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
```

Science and Technology Facilities Council

A complete example (part 3)

```
GOInvokeSchedule[invoke='invoke_0', Constant loop bounds=False]
    0: Directive[OMP parallel]
        Schedule[]
            0: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_ct', it_space='go_internal_pts']
                        Literal[value:'ssha_t%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ssha_t%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
                            0: Loop[type='inner', field_space='go_ct', it_space='go_internal_pts']
                                Literal[value:'ssha_t%internal%xstart', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'ssha_t%internal%xstop', Scalar<INTEGER, UNDEFINED>]
                                Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                                Schedule[]
                                    0: CodedKern continuity_code(ssha_t,sshn_t,sshn_u,sshn_v,hu,hv,un,vn,area_t)
            1: Directive[OMP do]
                Schedule[]
                    0: Loop[type='outer', field_space='go_cu', it_space='go_internal_pts']
                        Literal[value:'ua%internal%ystart', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'ua%internal%ystop', Scalar<INTEGER, UNDEFINED>]
                        Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                        Schedule[]
```

```
!$omp parallel default(shared), private(i,j)
!$omp do schedule(static)
DO j=ssha_t%internal%ystart,ssha_t%internal%ystop
  DO i=ssha_t%internal%xstart,ssha_t%internal%xstop
    CALL continuity_code(i, j, ssha_t%data, sshn_t%data, sshn_u%data, sshn_v%data, hu%data, hv%data, un%data, vn%data, sshn_t%grid%area_t)
  END DO
END DO
!$omp end do
!$omp do schedule(static)
DO j=ua%internal%ystart,ua%internal%ystop
```

Current capabilities overview

Frontend / Creation

- Direct PSyIR creation
- fparser Fortran
  (LFRic, GOcean and NEMO DSLs use this frontend)

Backends

- Fortran
  (most mature backend)
- OpenCL
  (only GOcean kernels)
- SIR
  (only NEMO)
- C
  (only to support OpenCL)

Take Away

- PSyIR is the Internal Representation at the core of PSyclone.
  - It is **mutable** to allow HPC experts to encode optimization steps by applying transformations or directly interacting with its API.
  - It is **language-independent** to allow the generation of output source-code in multiple programming languages.
  - A list of **Transformations** are provided for easy manipulation of the code.

Questions?

Read more about PSyIR at:
https://psyclone.readthedocs.io/en/latest/psyir.html

Contact: sergi.siso@stfc.ac.uk