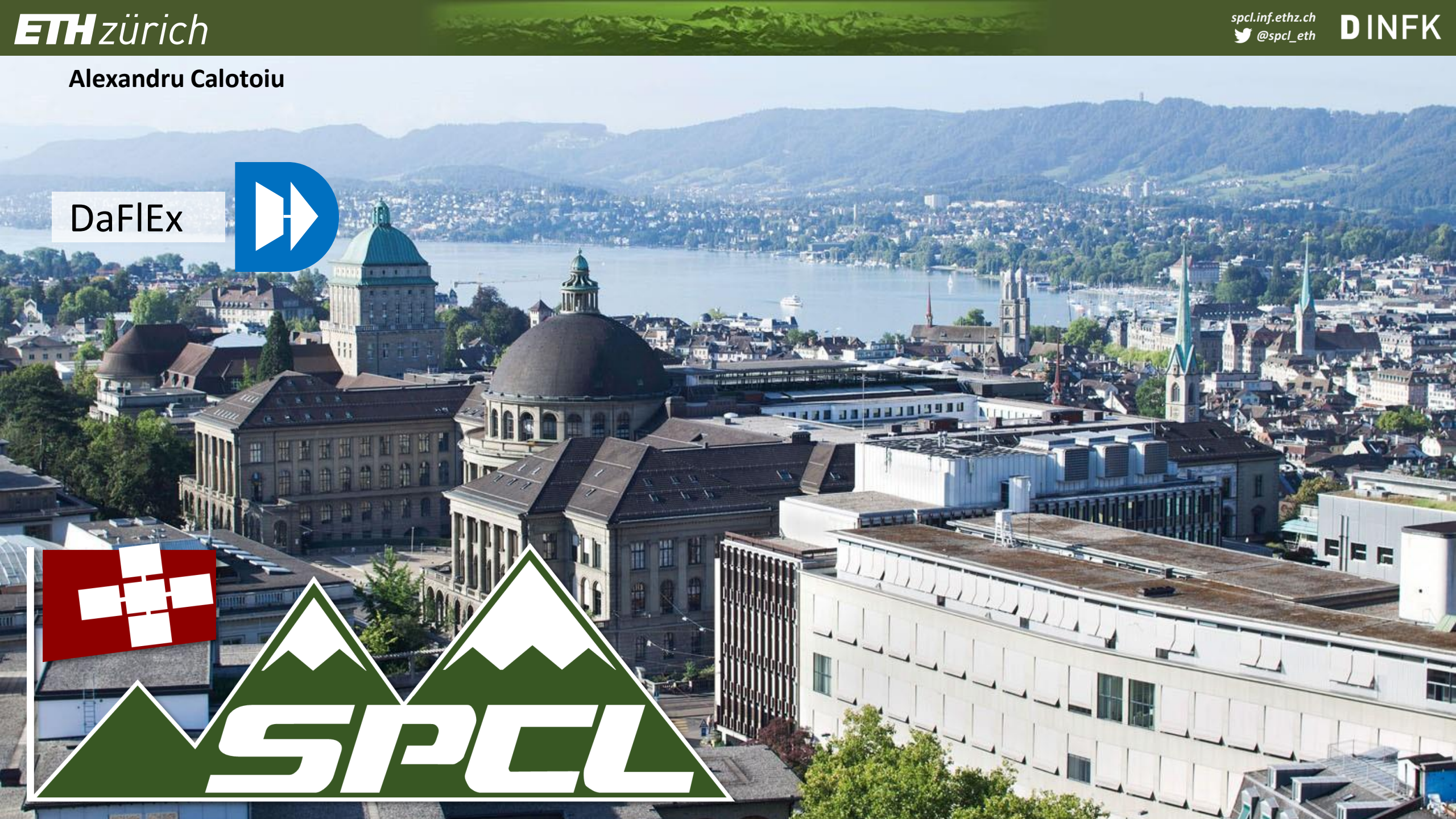
Alexandru Calotoiu

DaFlEx



F2DaCe

```
parser = ParserFactory().create(std="f2008")
reader = FortranFileReader(
    os.path.realpath("C:/Users/Alexwork/Desktop/Git/f2dace/tests/cloudsc_nostruct.f90"))
    #os.path.realpath("C:/Users/Alexwork/Desktop/Git/f2dace/tests/ifissue.f90"))
ast = parser(reader)
```

Create fparser AST

```
own_ast = create_own_ast(ast)
```

Create internal AST

```
own_ast = CallToArray(fd).visit(own_ast)
own_ast = CallToSRCall(justfd).visit(own_ast)
transformations = [
        CallExtractor,
        MoveReturnValueToArguments,
        IndicesExtractor,
        ReadWriteAdder,
        ]

for transformation in transformations:
        own_ast = transformation().visit(own_ast)
```

A(x)

Disambiguating CallExpression sillyness

Canonicalize AST

```
translator.translate(own_ast, globalsdfg)

#globalsdfg=SDFG.from_file('fortran_init.sdfg'
globalsdfg.save("fortran_init.sdfg")
globalsdfg.validate()
```

Create SDFG

The “const” pass

```
INTEGER, PARAMETER :: JPIM = SELECTED_INT_KIND(9)
INTEGER, PARAMETER :: JPIB = SELECTED_INT_KIND(12)

INTEGER, PARAMETER :: JPRB = SELECTED_REAL_KIND(6,37)

REAL(KIND=JPRB) :: RPI
```

There is a need to store all parameters before being able to even declare other variables!

The “read/written” pass

```
def visit_BinOp(self,node: BinOp):
    retnode=self.generic_visit(node)
    retnode.read_vars=list(set().union(retnode.lvalue.read_vars,retnode.rvalue.read_vars))
    if (retnode.op == "="):
        retnode.written_vars=[retnode.lvalue.name]
    else:
        if hasattr(retnode.lvalue,"name"):
            if retnode.lvalue.name not in retnode.read_vars:
                retnode.read_vars.append(retnode.lvalue.name)
    if hasattr(retnode.rvalue,"name"):
        if retnode.rvalue.name not in retnode.read_vars:
            retnode.read_vars.append(retnode.rvalue.name)
    else:
        if hasattr(retnode.rvalue,"read_vars"):
            for i in retnode.rvalue.read_vars:
                retnode.read_vars.append(i)
    return retnode
```

Recursively create list of all read/written variables in all context – append the AST

Function statements

```
class replaceStatementFunctionPass(NodeTransformer):

    def __init__(self, statefunc: list):
        self.funcs = statefunc

    def visit_StructureConstructor(self, node: StructureConstructor):
        for i in self.funcs:
            if node.name == i[0].name:
                ret_node = copy.deepcopy(i[1])
                ret_node = renameArguments(node.args,
                                           i[0].args).visit(ret_node)
                return ParenExpr(expr=ret_node)
        return self.generic_visit(node)

    def visit_CallExpr(self, node: CallExpr):
        for i in self.funcs:
            if node.name == i[0].name:
                ret_node = copy.deepcopy(i[1])
                ret_node = renameArguments(node.args,
                                           i[0].args).visit(ret_node)
                return ParenExpr(expr=ret_node)
        return self.generic_visit(node)
```

```
implicit none
INTEGER :: AR(3)
INTEGER :: IDX =1
INTEGER :: FUNC
FUNC(IDX)=IDX*2
AR(IDX)=IDX*2
AR(2)=5
write (*,*) AR(1), AR(2), FUNC(2)
end
```

Structures

```
type state_type
  !$loki dimension(klon,klev)
  REAL(KIND=JPRB), dimension(:,:), pointer :: u,v,T   ! GMV fields
  !$loki dimension(klon,klev)
  REAL(KIND=JPRB), dimension(:,:), pointer :: o3,q,a  ! GFL fields
  !$loki dimension(klon,klev,5)
  REAL(KIND=JPRB), dimension(:,:,:), pointer :: cld   ! composed cloud array
  !REAL(KIND=JPRB), dimension(:,:), pointer :: qsat    ! spec. humidity at saturation
end type state_type
```

```
639    TYPE (STATE_TYPE) , INTENT (IN)  :: tendency_cml   ! cumulative tendency used for final output
640    TYPE (STATE_TYPE) , INTENT (IN)  :: tendency_tmp   ! cumulative tendency used as input
641    TYPE (STATE_TYPE) , INTENT (OUT) :: tendency_loc   ! local tendency from cloud scheme
```

```
REAL(KIND=JPRB) :: tendency_cml_u(KLON,KLEV,NBLOCKS) ,tendency_cml_v(KLON,KLEV,NBLOCKS) ,tendency_cml_T(KLON,KLEV,NBLOCKS)    ! GMV fields
REAL(KIND=JPRB):: tendency_cml_o3(KLON,KLEV,NBLOCKS) ,tendency_cml_q(KLON,KLEV,NBLOCKS) ,tendency_cml_a(KLON,KLEV,NBLOCKS)   ! GFL fields
REAL(KIND=JPRB):: tendency_cml_cld(KLON,KLEV,NCLV,NBLOCKS)    ! composed cloud array

REAL(KIND=JPRB):: tendency_tmp_u(KLON,KLEV,NBLOCKS),tendency_tmp_v(KLON,KLEV,NBLOCKS),tendency_tmp_T(KLON,KLEV,NBLOCKS)   ! GMV fields
REAL(KIND=JPRB):: tendency_tmp_o3(KLON,KLEV,NBLOCKS),tendency_tmp_q(KLON,KLEV,NBLOCKS),tendency_tmp_a(KLON,KLEV,NBLOCKS)  ! GFL fields
REAL(KIND=JPRB)  :: tendency_tmp_cld(KLON,KLEV,NCLV,NBLOCKS)   ! composed cloud array

REAL(KIND=JPRB):: tendency_loc_u(KLON,KLEV,NBLOCKS),tendency_loc_v(KLON,KLEV,NBLOCKS),tendency_loc_T(KLON,KLEV,NBLOCKS)   ! GMV fields
REAL(KIND=JPRB):: tendency_loc_o3(KLON,KLEV,NBLOCKS),tendency_loc_q(KLON,KLEV,NBLOCKS),tendency_loc_a(KLON,KLEV,NBLOCKS)  ! GFL fields
REAL(KIND=JPRB) :: tendency_loc_cld(KLON,KLEV,NCLV,NBLOCKS)   ! composed cloud array
```