Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

ICON-DSL Overview

C. Müller, M. Röthlin, G. Serafini, C. Osuna, B. Weber

COSMO GM, 07.09.2021

Motivation

Model software development starts at numerical discretization of continuous quantities:

$$\underline{\nabla}_{\underline{n}}\psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

Motivation

- (very) straight forward implementation
- "actual science" + mesh

```
DO jk = slev, elev
 DO je = i_startidx, i_endidx
   grad_norm_psi_e(je,jk) =
      (psi_c(iidx(je,2),jk)-psi_c(iidx(je,1),jk))/lhat(je)
 ENDDO
END DO
```

Motivation

- turns out mesh is too large for one machine, add blocks

```
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, jb, i_startblk, i_endblk, &
                    i_startidx, i_endidx, rl_start, rl_end)
 DO jk = slev, elev
   DO je = i_startidx, i_endidx
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
```

Motivation

- code doesn't perform, add directives to exploit shared memory machines

```
#ifdef _OMP
!$OMP PARALLEL
!$OMP DO PRIVATE(jb, i_startidx, i_endidx, je, jk)
#endif
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, jb, i_startblk, i_endblk, &
                    i_startidx, i_endidx, rl_start, rl_end)
 DO jk = slev, elev
   DO je = i_startidx, i_endidx
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
#ifdef _OMP
!$OMP END DO NOWAIT
!$OMP END PARALLEL
#endif
```

Motivation

- code needs to target another architecture...
- ... with different optimal memory layout

```
#ifdef _OMP
!$OMP ....
#else
!$ACC ....
#endif
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 #ifdef __LOOP_EXCHANGE
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
 #else
   DO jk = slev, elev
     DO je = i_startidx, i_endidx
 #endif
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
#ifdef _OMP
!$OMP ...
#else
!$ACC ...
#endif
```

Motivation

$$\underline{\nabla_n}\psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

```
#ifdef _OMP
!$OMP ....
#else
!$ACC ....
#endif
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 #ifdef __LOOP_EXCHANGE
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
 #else
   DO jk = slev, elev
     DO je = i_startidx, i_endidx
 #endif
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
#ifdef _OMP
!$OMP ...
#else
!$ACC ...
#endif
```

MeteoSwiss

Motivation

What if

- Requirements change, e.g. it turns out that this gradient should have been approximated using a higher order stencil?
- A third (fourth...) architecture needs to be supported?
- The mesh library needs to be replaced?
- Fusion of stencils?

```
#ifdef _OMP
!$OMP ....
#else
!$ACC ....
#endif
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 #ifdef __LOOP_EXCHANGE
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
 #else
   DO jk = slev, elev
     DO je = i_startidx, i_endidx
 #endif
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
#ifdef _OMP
!$OMP ...
#else
!$ACC ...
#endif
```

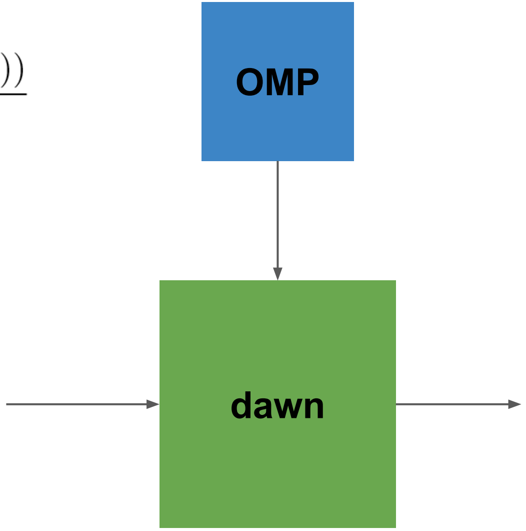
Motivation

No FORTRAN Backend Exists, only for illustration purposes

Idea of DSLs in general

$$\underline{\nabla_n}\psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

```
grad_norm_psi_e =
     sum_over(psi_c,
              Edge > Cell,
              [1/lhat, -1/lhat]
)
```



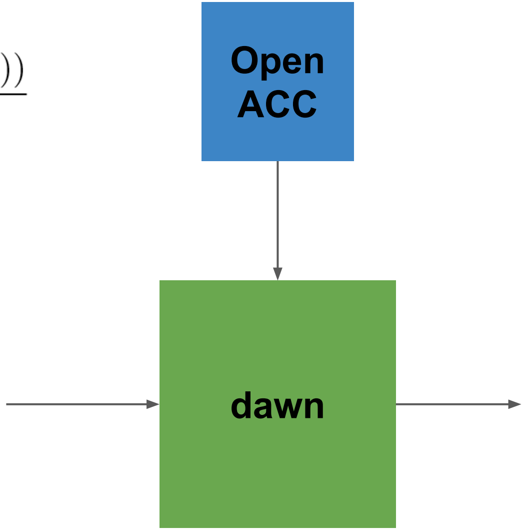
```
!$OMP PARALLEL
!$OMP DO PRIVATE(jb, i_startidx, i_endidx, je, jk)
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

Motivation

Idea of DSLs in general

$$\underline{\nabla_n}\psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

```
grad_norm_psi_e =
     sum_over(psi_c,
              Edge > Cell,
              [1/lhat, -1/lhat]
)
```



No FORTRAN Backend Exists, only for illustration purposes

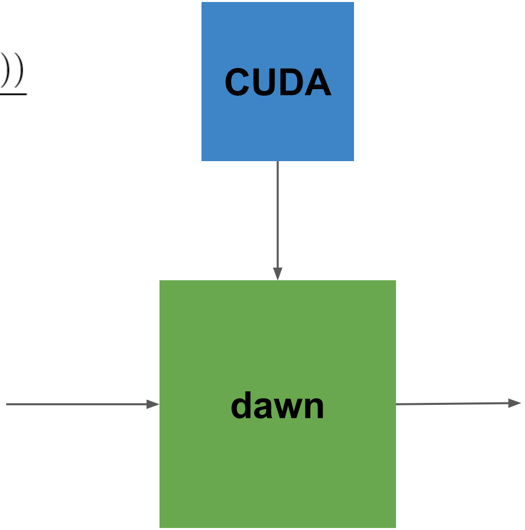
```
!$ACC PARALLEL &
!$ACC PRESENT(ptr_patch, iidx, iblk, pci_c, grad_...)
!$ACC LOOP GANG
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 DO jk = slev, elev
   DO je = i_startidx, i_endidx
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )
       / ptr_patch%edges%lhat(je,jb)
   ENDDO
 END DO
END DO
!$ACC END PARALLEL
!$ACC END DATA
```

Motivation

Idea of DSLs in general

$$\nabla_n \psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

```
grad_norm_psi_e =
      sum_over(psi_c,
               Edge > Cell,
               [1/lhat, -1/lhat]
)
```



CUDA

dawn

```
unsigned int pidx = blockIdx.x * blockDim.x + threadIdx.x;
unsigned int kidx = blockIdx.y * blockDim.y + threadIdx.y;
int klo = kidx * LEVELS PER THREAD;
int khi = (kidx + 1) * LEVELS_PER_THREAD;
if(pidx >= hSize) {
 return;
}
for(int kIter = klo; kIter < khi; kIter++) {
 if(kIter >= kSize) {
   return;
 }
 ::dawn::float type lhs 62 = (::dawn::float type)0;
 ::dawn::float type weights_62[2] = {((::dawn::float_type)1.0
/ globals.lhat),
                                    ((::dawn::float_type)-1.0
/ globals.lhat)};
 for(int nbhIter = 0; nbhIter < E C SIZE; nbhIter++) {
   int nbhIdx = ecTable[pidx * E C SIZE + nbhIter];
   if(nbhIdx == DEVICE_MISSING_VALUE) {
     continue;
   }
   lhs_62 += weights_62[nbhIter] * psi_c[kIter * CellStride +
nbhIdx];
 }
 grad_normal_psi_e[kIter * EdgeStride + pidx] = lhs_62;
```

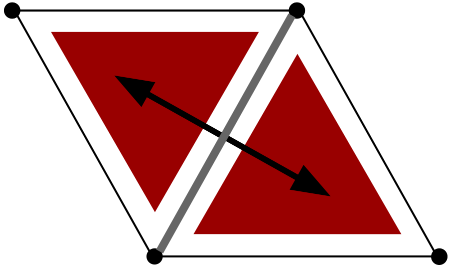
Dusk notation

$$\underline{\nabla_n}\psi(e) = \frac{\psi(c_1(e)) - \psi(c_0(e))}{\hat{l}}$$

```
grad_norm_psi_e =
     sum_over(psi_c, Edge > Cell, [1/lhat, -1/lhat])
```

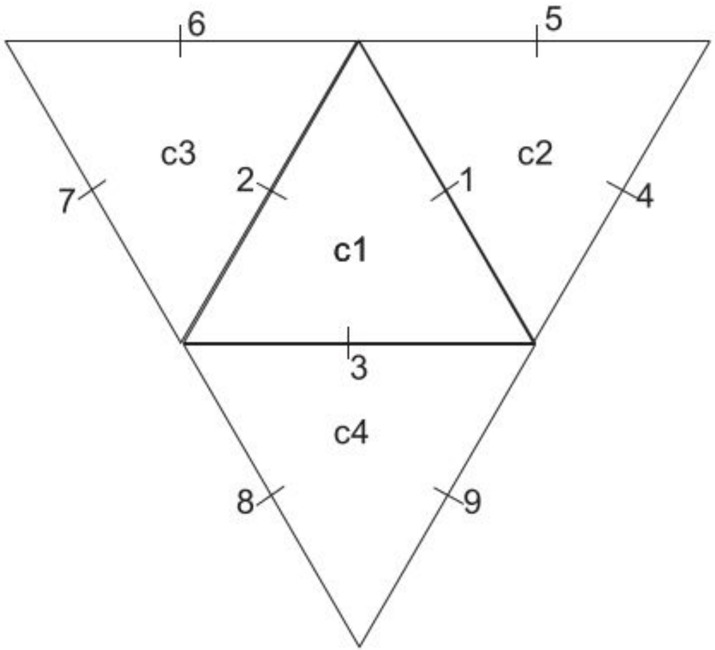
Neighborhood Chain:

```
Edge > Cell
```



Dusk notation - Neighbor Chains

Neighborhood selection as a "first class citizen" of the language design



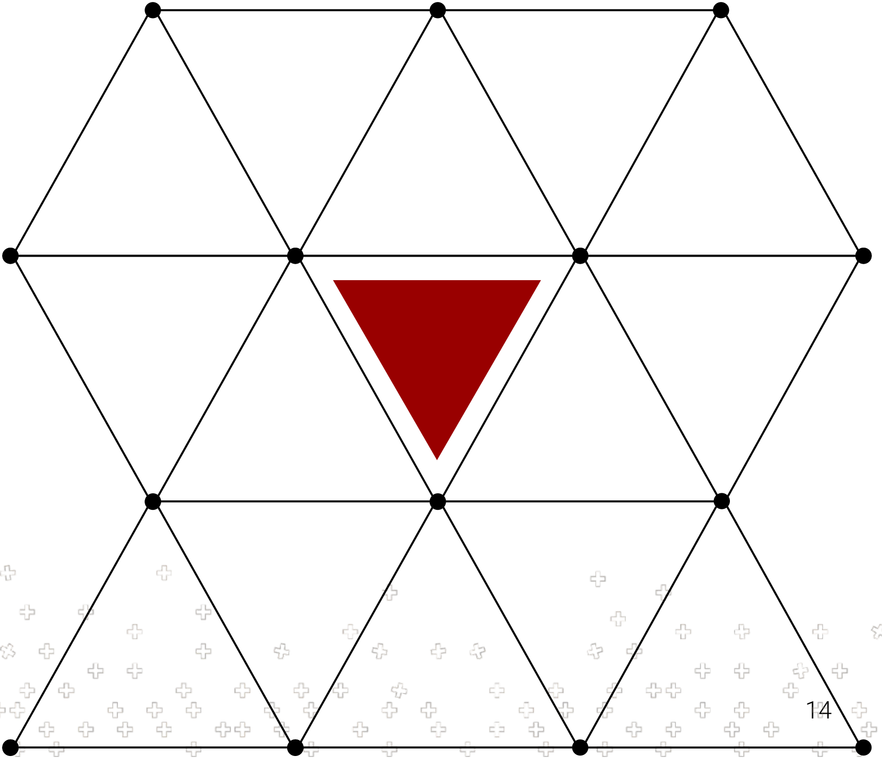
$$f(c_1) = \sum_{j=1}^{9} f_j w_j$$

```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```

The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core - Zängl et al

Dusk notation - Neighbor Chains

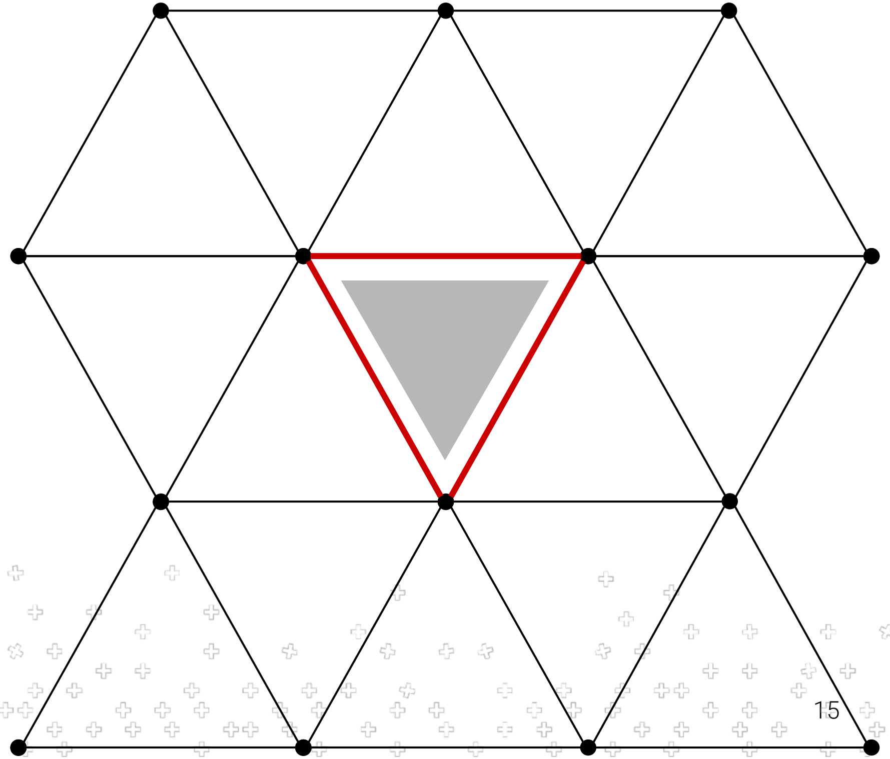
```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
        fc = sum_over(Cell > Edge > Cell > Edge,
                      w*fe)
```



MeteoSwiss

Dusk notation - Neighbor Chains

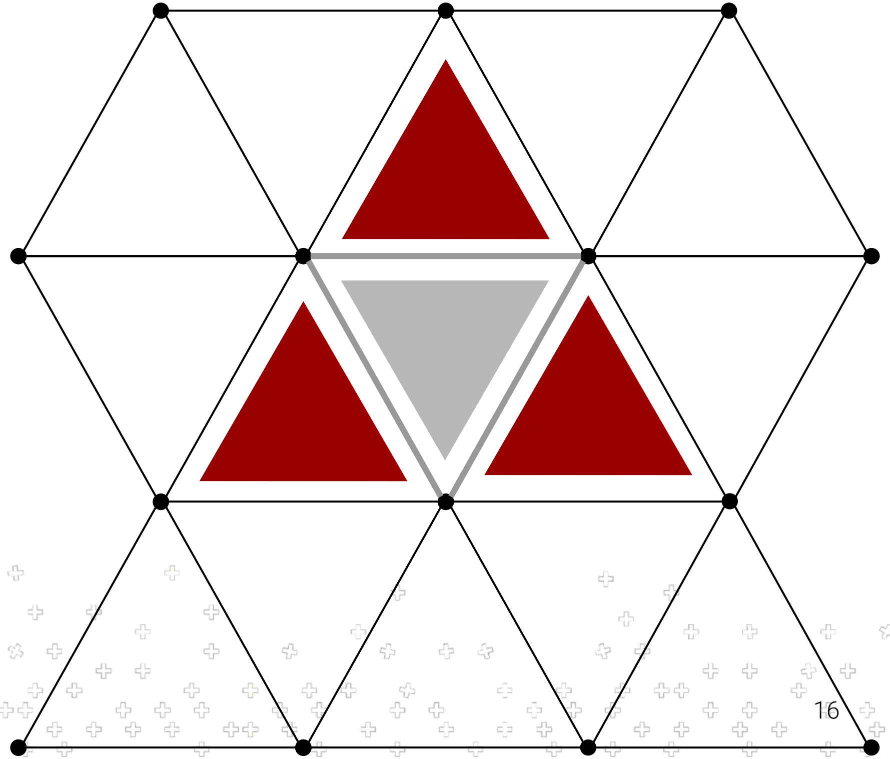
```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```



MeteoSwiss

Dusk notation - Neighbor Chains

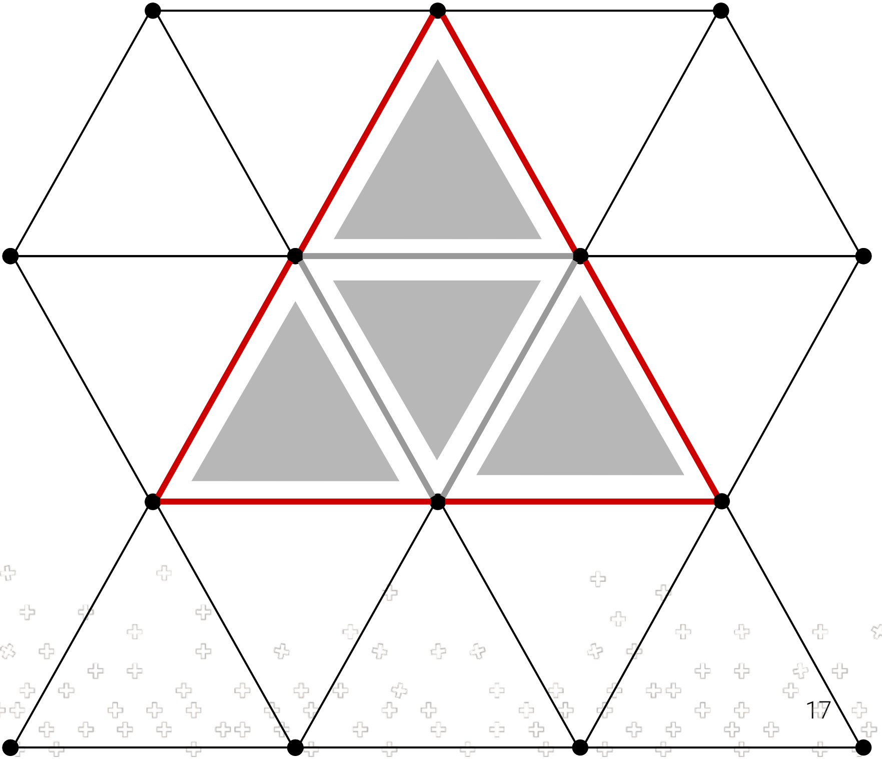
```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```



MeteoSwiss

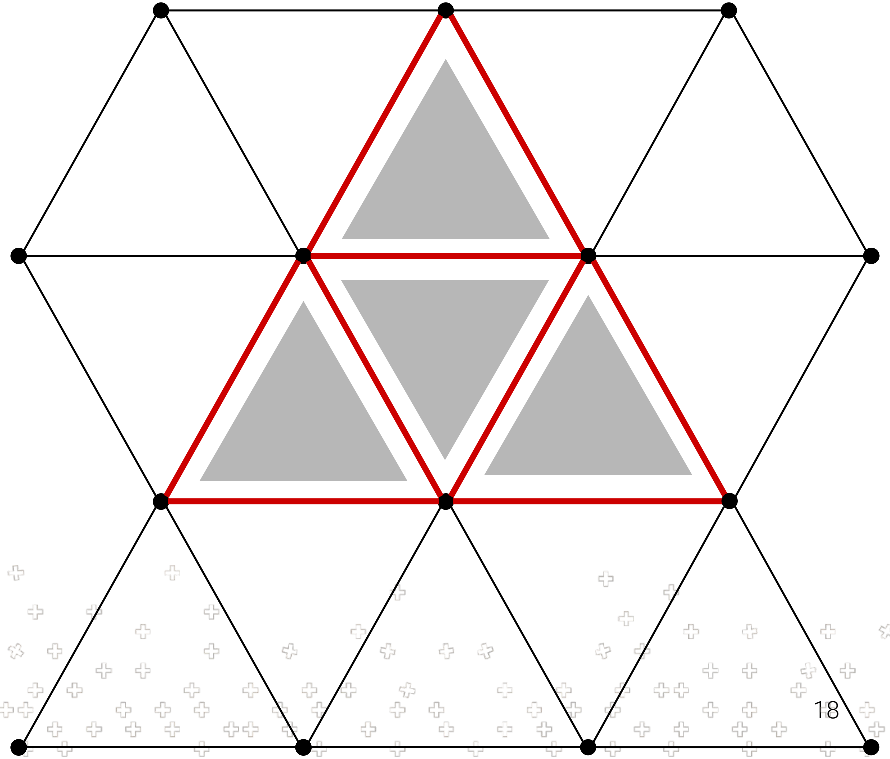
Dusk notation - Neighbor Chains

```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```



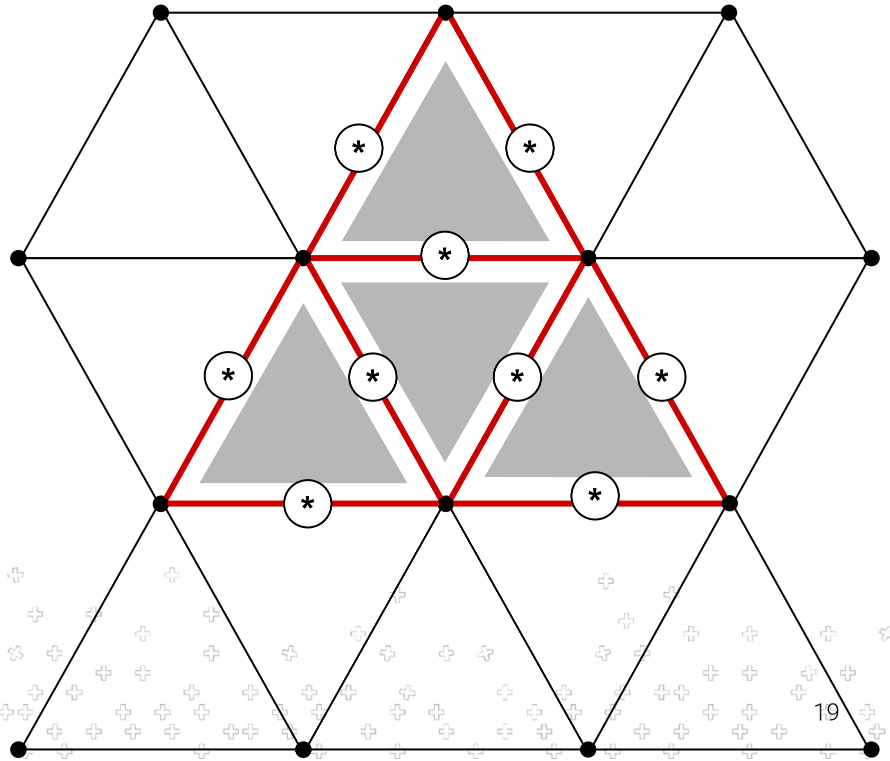
Dusk notation - Neighbor Chains

```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
        fc = sum_over(Cell > Edge > Cell > Edge,
                      w*fe)
```



Dusk notation - Neighbor Chains

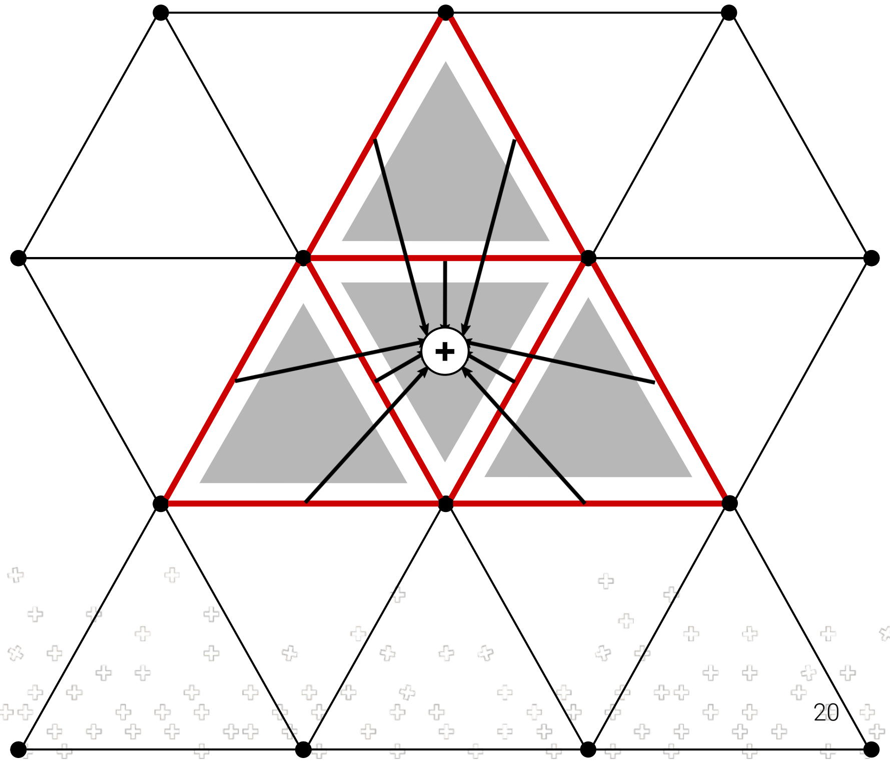
```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```



MeteoSwiss

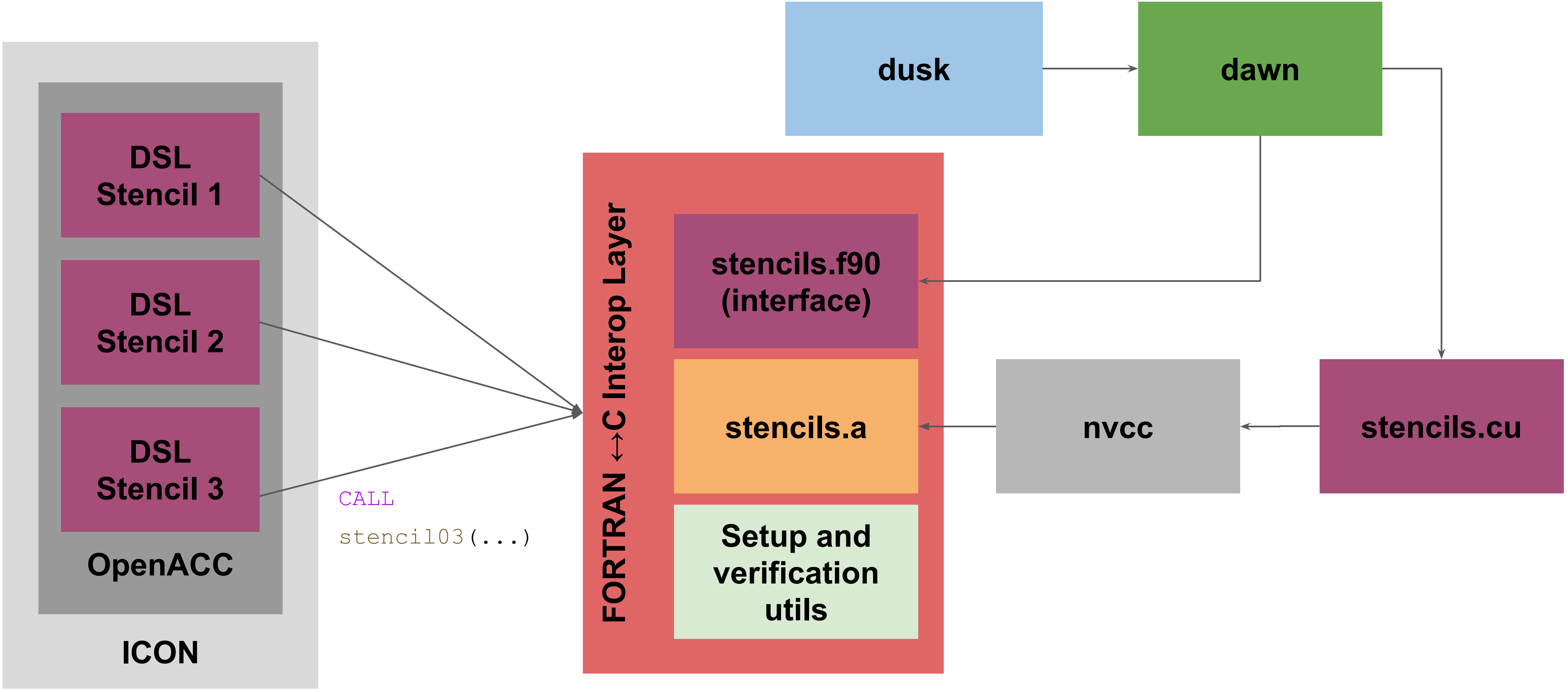
Dusk notation - Neighbor Chains

```
@stencil
def intp(fc: Field[Cell],
         fe: Field[Edge],
         w: Field[Cell > Edge > Cell > Edge]):
   with levels_downward:
       fc = sum_over(Cell > Edge > Cell > Edge,
                     w*fe)
```



MeteoSwiss

Interoperability



```
CALL
stencil03(...)
```

MeteoSwiss

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)

...



         ! Lateral boundary diffusion for vn
     i_startblk = p_patch%edges%start_block(start_bdydiff_e)
     i_endblk   = p_patch%edges%end_block(grf_bdywidth_e)

!$OMP DO PRIVATE(je,jk,jb,i_startidx,i_endidx) ICON_OMP_DEFAULT_SCHEDULE
     DO jb = i_startblk,i_endblk

       CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                          i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

!$ACC PARALLEL LOOP DEFAULT(NONE) GANG VECTOR COLLAPSE(2) ASYNC(1) IF( i_am_accel_node .AND. acc_on
)
       DO jk = 1, nlev
!DIR$ IVDEP
         DO je = i_startidx, i_endidx
           p_nh_prog%vn(je,jk,jb) =    &
             p_nh_prog%vn(je,jk,jb) + &
             z_nabla2_e(je,jk,jb) * &
             p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
         ENDDO
       ENDDO
     ENDDO
!$OMP END DO
```

MeteoSwiss

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)
```

```
DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                    i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)




DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                  i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

```
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```

```fortran
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)

DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                   i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

```python
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[ lb+4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)




DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                   i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

```
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)




DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                    i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

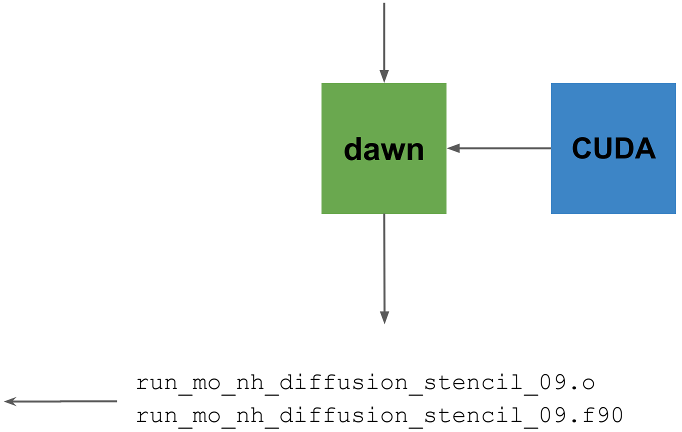
```
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)





DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                   i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

```
fac_bdydiff_v = Global("fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge *   fac_bdydiff_v
```

```
rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)




DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                   i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
     p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
```

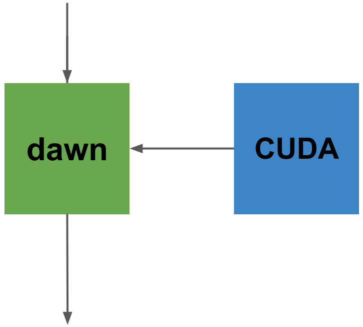
```
fac_bdydiff_v = Global("fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K],
   area_edge: Field[Edge],
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```



```
run_mo_nh_diffusion_stencil_09.o
run_mo_nh_diffusion_stencil_09.f90
```

```
!rl_start = start_bdydiff_e
!rl_end    = grf_bdywidth_e
!
!i_startblk = p_patch%edges%start_block(rl_start)
!i_endblk    = p_patch%edges%end_block(rl_end)
!
!
!
!
!
!DO jb = i_startblk,i_endblk
!    CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
!            i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)
!
!    DO jk = 1, nlev
!        DO je = i_startidx, i_endidx
!            p_nh_prog%vn(je,jk,jb) = p_nh_prog%vn(je,jk,jb) + z_nabla2_e(je,jk,jb) * &
!                    p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
!        ENDDO
!    ENDDO
!ENDDO
```

```
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```



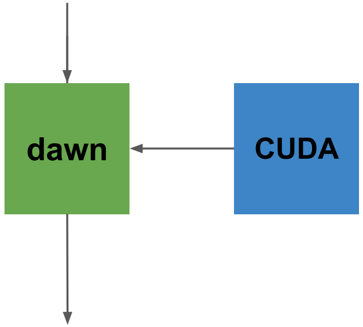
```
run_mo_nh_diffusion_stencil_09.o
run_mo_nh_diffusion_stencil_09.f90
```

```
CALL run_mo_nh_diffusion_stencil_09(fac_bdydiff_v, z_nabla2_e(:,:,1), &
    p_patch%edges%area_edge(:,1), p_nh_prog%vn(:,:,1), p_nh_prog_vn_before(:,:,1))
```

MeteoSwiss

```
#ifdef __DSL_VERIFY
!$ACC PARALLEL
p_nh_prog_vn_before(:,:,:) = p_nh_prog%vn(:,:,:)
!$ACC END PARALLEL

rl_start = start_bdydiff_e
rl_end   = grf_bdywidth_e

i_startblk = p_patch%edges%start_block(rl_start)
i_endblk   = p_patch%edges%end_block(rl_end)

DO jb = i_startblk,i_endblk
 CALL get_indices_e(p_patch, jb, i_startblk, i_endblk,&
                    i_startidx, i_endidx, start_bdydiff_e, grf_bdywidth_e)

 DO jk = 1, nlev
   DO je = i_startidx, i_endidx
     p_nh_prog%vn(je,jk,jb) =  &
       p_nh_prog%vn(je,jk,jb) +&
       z_nabla2_e(je,jk,jb) *&
       p_patch%edges%area_edge(je,jb)*fac_bdydiff_v
   ENDDO
 ENDDO
ENDDO
#endif
CALL wrap_run_mo_nh_diffusion_stencil_09(fac_bdydiff_v, z_nabla2_e(:,:,1), &
    p_patch%edges%area_edge(:,1), p_nh_prog%vn(:,:,1), p_nh_prog_vn_before(:,:,1))
```

```
fac_bdydiff_v = Global( "fac_bdydiff_v")

@stencil
def mo_nh_diffusion_stencil_09 (
   z_nabla2_e: Field[Edge, K] ,
   area_edge: Field[Edge] ,
   p_nh_prog_vn: Field[Edge, K]
):
   with domain.upward.across[lb+ 4:nudging-1]:
       p_nh_prog_vn += z_nabla2_e * area_edge * fac_bdydiff_v
```



wrap_run_mo_nh_diffusion_stencil_09.f90

run_mo_nh_diffusion_stencil_09.f90

run_and_verify_mo_nh_diffusion_stencil_09.f90

Diffusion module translation

- All 16 stencils translated on our code path
  - `diffu_type == 5`
  - `l_limited_area == .TRUE.`
  - `nblks == 1`
  - …

Diffusion module translation

- All 16 stencils translated on our code path
  - `diffu_type == 5`
  - `l_limited_area == .TRUE.`
  - `nblks == 1`
  - …

- Still in Fortran
  - Stencils not in our code path

Diffusion module translation

- All 16 stencils translated on our code path
  - `diffu_type == 5`
  - `l_limited_area == .TRUE.`
  - `nblks == 1`
  - …

- Still in Fortran
  - Stencils not in our code path
  - **Synchronization**

```
CALL sync_patch_array (SYNC_E, p_patch, p_nh_prog%vn,opt_varname="diffusion: vn
sync")
```

```
IF (diffu_type == 3) THEN  ! Only Smagorinsky diffusion
 IF ( jg == 1 .AND. l_limited_area .OR. jg > 1 .AND. .NOT. lfeedback(jg)) THEN
  ...
 ENDIF
ENDIF
```

Diffusion module translation

- All 16 stencils translated on our code path
  - `diffu_type == 5`
  - `l_limited_area == .TRUE.`
  - `nblks == 1`
  - …

- Still in Fortran
  - Stencils not in our code path
  - Synchronization
  - **Control flow**

```
CALL sync_patch_array(SYNC_E, p_patch, p_nh_prog%vn,opt_varname="diffusion: vn
sync")
```

```
IF (diffu_type == 3) THEN ! Only Smagorinsky diffusion
 IF ( jg == 1 .AND. l_limited_area .OR. jg > 1 .AND. .NOT. lfeedback(jg)) THEN
  ...
 ENDIF
ENDIF
```

Diffusion module translation

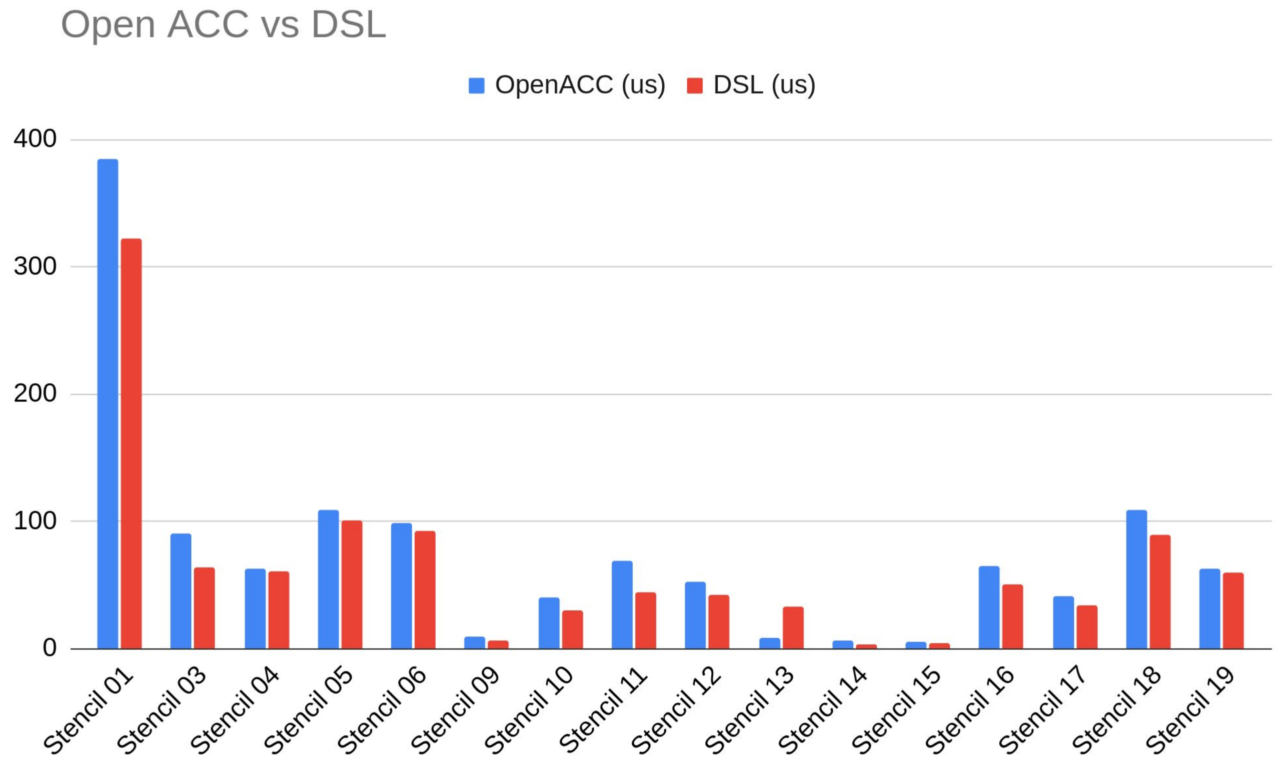
- All 16 stencils translated on our code path
  - `diffu_type == 5`
  - `l_limited_area == .TRUE.`
  - `nblks == 1`
  - …

- Still in Fortran
  - Stencils not in our code path
  - Synchronization
  - Control flow

```
CALL sync_patch_array (SYNC_E, p_patch, p_nh_prog%vn,opt_varname="diffusion: vn sync")
```

```
IF (diffu_type == 3) THEN ! Only Smagorinsky diffusion
 IF ( jg == 1 .AND. l_limited_area .OR. jg > 1 .AND. .NOT. lfeedback(jg)) THEN
  ...
 ENDIF
ENDIF
```

Diffusion module translation - Performance



MeteoSwiss

Diffusion module translation - Performance

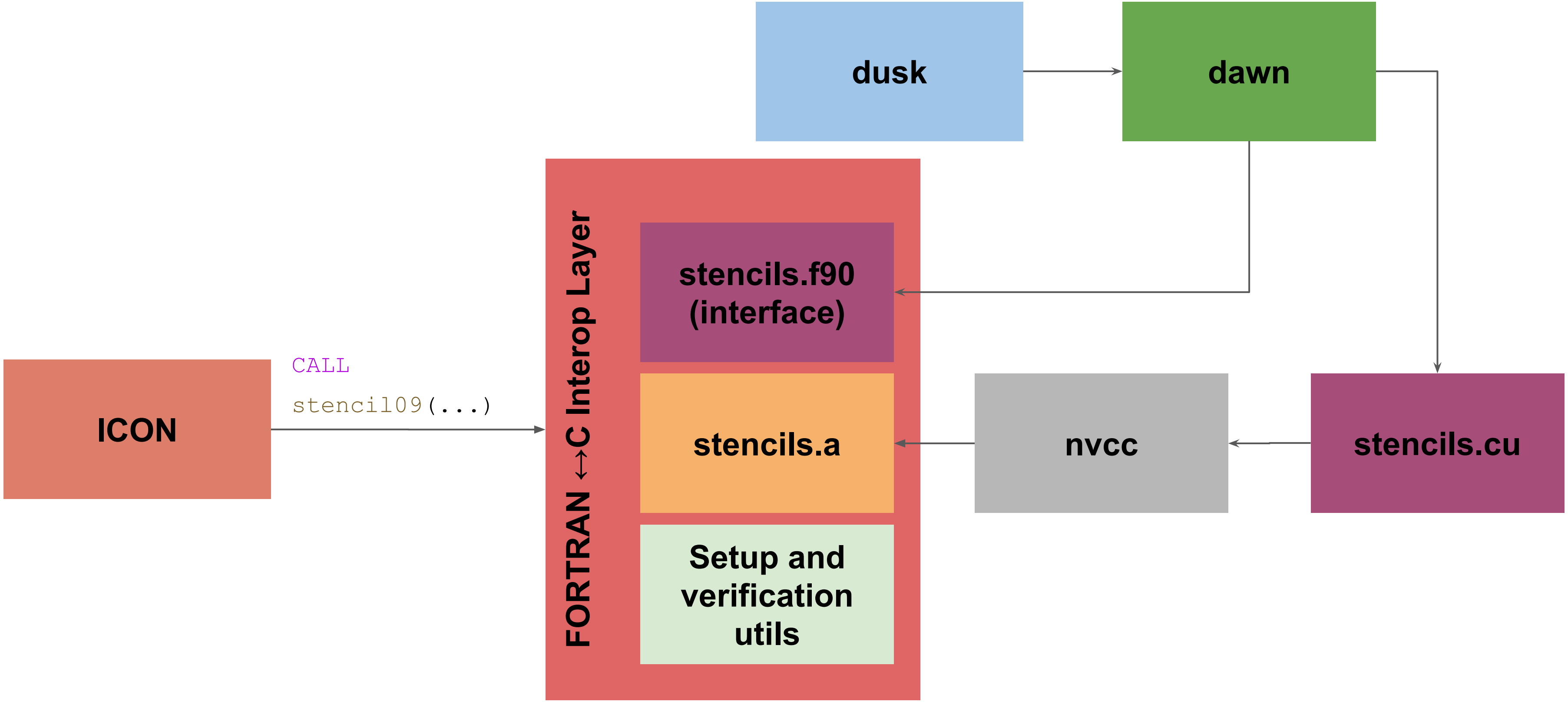
- All DSL stencils outperform all OpenACC stencils, except one (stencil 13)
- Performance increase ranges from 6% to 87% (excluding stencil 13)
- Average performance increase is 23%

- Further performance increase expected due to
  - fusing of kernels
  - more involved inlining passes (e.g. reduction inlining)

Dycore Translation - Progress

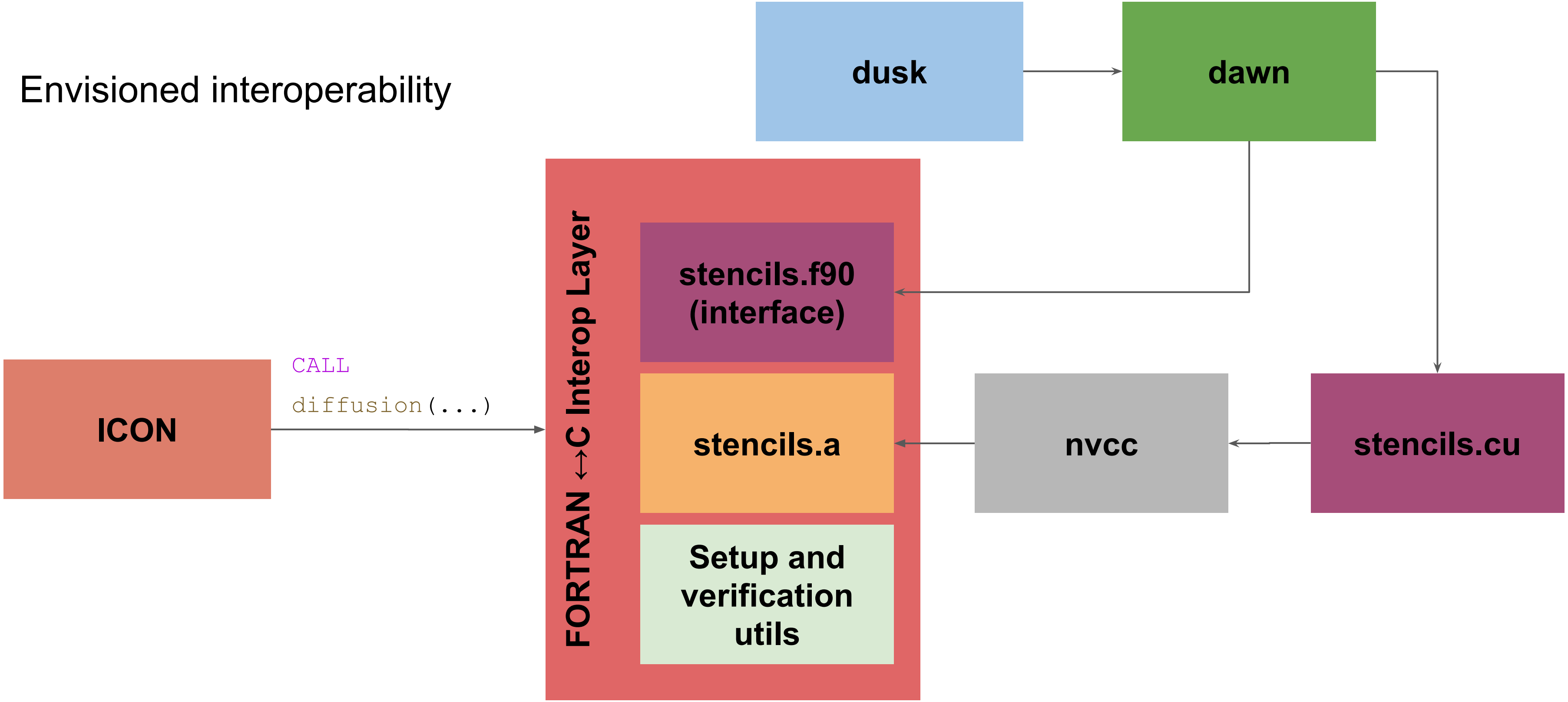
Module	**Status**
mo_nh_diffusion	Integrated in ICON, NWP verifies, some performance optimization
mo_solve_nonhydro	Integrated in ICON, NWP verifies.
mo_velocity_advection	Integrated in ICON, NWP verifies, under review

Interoperability



Interoperability

Envisioned interoperability



Summary

- Status:
  - 16 stencils in diffusion translated
  - Integrated
  - Verified

- Outlook:
  - Profiling and Optimizations
  - DSL’ify control flow and Synchronizations
  - Translate dycore

- Code available:
  - https://gitlab.dkrz.de/dsl/icon-cscs/-/tree/add_DSL/dsl

Additional slides

Fusion

- Three stencils, in divergent control flow
  - How to fuse?

```
stencil0

IF (diffu_type == 3) THEN ! Only Smagorinsky diffusion
  stencil_1
ELSE
 stencil_2
ENDIF
```

Fusion

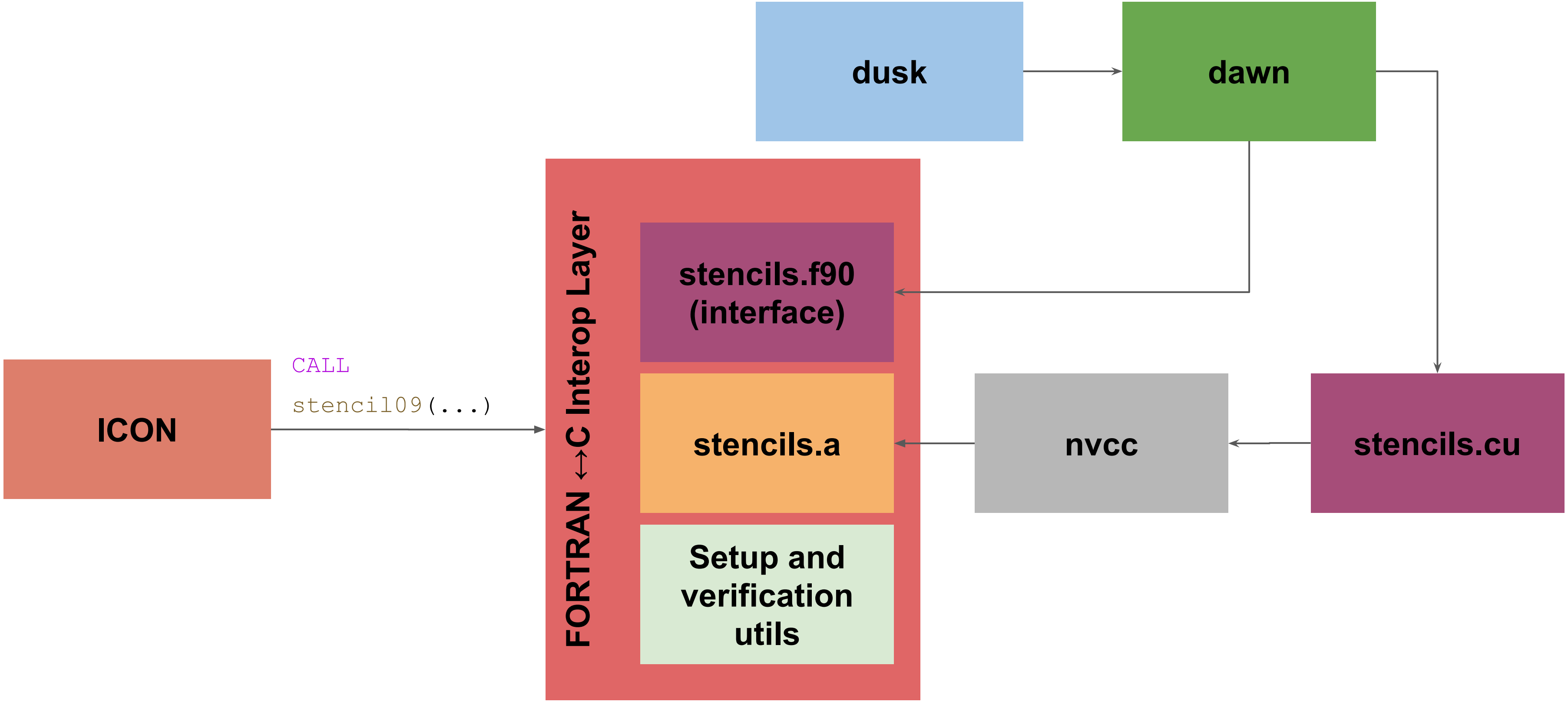
- Three stencils, in divergent control flow
    - How to fuse?

```
stencil0

IF (diffu_type == 3) THEN ! Only Smagorinsky diffusion
  stencil_1
ELSE
 stencil_2
ENDIF
```

- Fuse stencils 0+1 and 0+2:

```
IF (diffu_type == 3) THEN ! Only Smagorinsky diffusion
  fused_stencil_0_1
ELSE
  fused_stencil_0_2
ENDIF
```

Interoperability


dusk
dawn
FORTRAN ↔C Interop Layer
stencils.f90 (interface)
stencils.a
Setup and verification utils
ICON
CALL stencil09(...)
nvcc
stencils.cu