Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

Introduction to Domain Specific Languages

ESIWACE2 Summer School, 23rd August 2021

Speaker: Matthias Röthlin, MeteoSwiss

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

Contents

- What is a DSL?
- What is the purpose of a DSL (In the NWP context)?
- dawn as a NWP DSL case study
- Conclusions

Domain Specific Languages

What is a **D**omain **S**pecific **L**anguage?

- Transpose a Matrix in a general purpose Language (C) and in a DSL for matrix computations (Matlab)

Matlab

```
A = A'
```

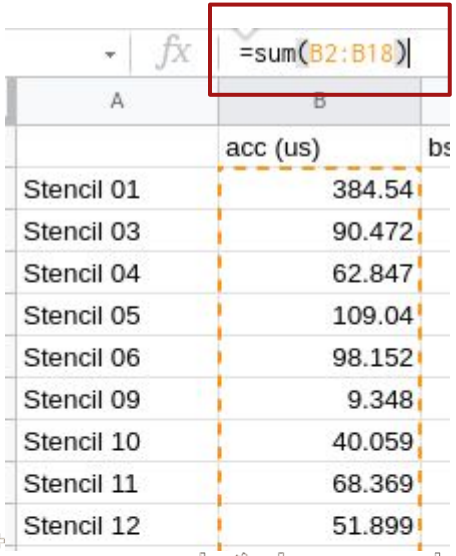
C

```
for (i = 1; i < N; i++) {
   for (j = 0; j < i; j++) {
        double tmp = A[i][j];
        A[i][j] = A[j][i];
        A[j][i] = tmp;
   }
}
```

Domain Specific Languages

What is a **D**omain **S**pecific **L**anguage?

- Spreadsheet Formula Evaluator



Domain Specific Languages

What is the purpose of a **D**omain **S**pecific **L**anguage?

- Matlab
  - Rapid Prototyping
- Spreadsheet Formula Evaluator
  - Finance / Spreadsheet Calculations
- NWP?

DSL Motivation for NWP

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

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

```
#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
```

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?

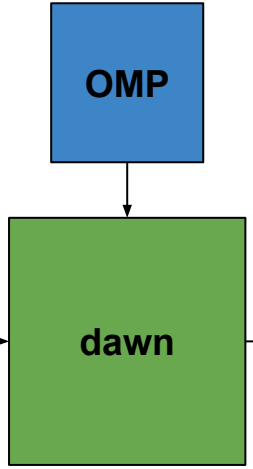
```
#ifdef _OMP
!$OMP ....
#else
!$ACC ....
#endif
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 #ifdef __LOOP_EXCHANGE
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
 #else
   DO jk = slev, elev
     DO je = i_startidx, i_endidx
 #endif
     grad_norm_psi_e(je,jk,jb) =  &
       ( 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
```

dawn Toolchain

Idea of dawn / DSLs in general

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

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



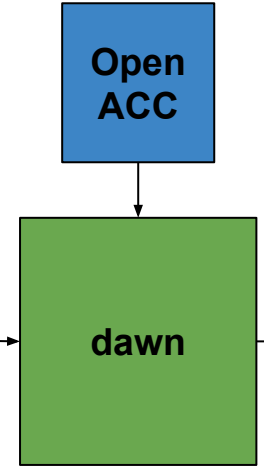

```
!$OMP PARALLEL
!$OMP DO PRIVATE(jb, i_startidx, i_endidx, je, jk)
DO jb = i_startblk, i_endblk
 CALL get_indices_e(ptr_patch, ...)
 DO je = i_startidx, i_endidx
   DO jk = slev, elev
     grad_norm_psi_e(je,jk,jb) =  &
       ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) -
         psi_c(iidx(je,jb,1),jk,iblk(je,jb,1))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

dawn Toolchain

Idea of dawn / DSLs in general

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

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




```
!$ACC PARALLEL &
!$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
```

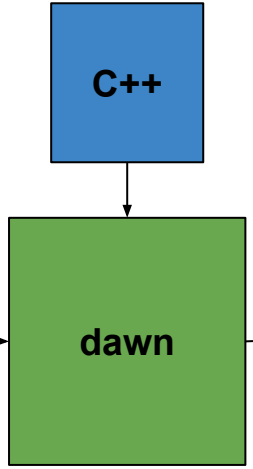
dawn Toolchain

No FORTRAN Backend Exists, only for illustration purposes

Idea of dawn / DSLs in general

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

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




```
for(int k = 0 + 0; k < m_k_size; ++k) {
 for(auto const& loc : getEdges(LibTag{}, m_mesh)) {
   for(auto inner_loc :
     grad_norm_psi_e(loc, k + 0) = reduce(
          LibTag{}, m_mesh, loc, (::dawn::float_type)0.0,
          std::vector<dawn::LocationType>
          {dawn::Edges, dawn::Cells},
          [&](auto& lhs, auto red_loc1, auto const& weight)
          {
            lhs += weight *psi_c(red_loc1, k + 0);
           return lhs;
          },
         std::vector<::dawn::float_type>({-1.0 ,1.0});
   }
   grad_norm_psi_e(loc, k + 0) /= lhat_e(loc, k + 0)
 }
}
```

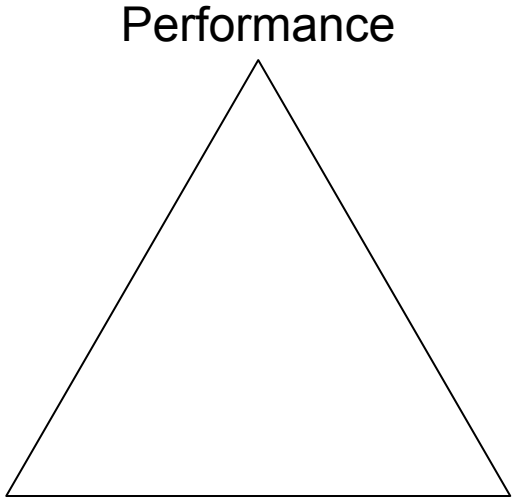
Domain Specific Languages

What is the purpose of a **D**omain **S**pecific **L**anguage?

- Matlab
  - Rapid Prototyping
- Spreadsheet Formula Evaluator
  - Finance / Spreadsheet Calculations
- NWP
  - Address the "3Ps"

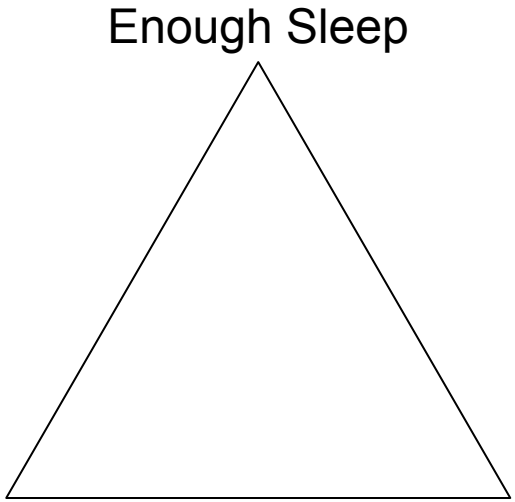
Performance, Portability, Productivity

Chose two of….



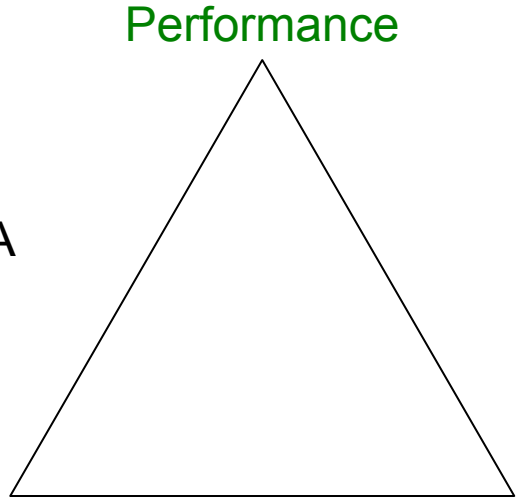
Students Dilemma

Chose two of….



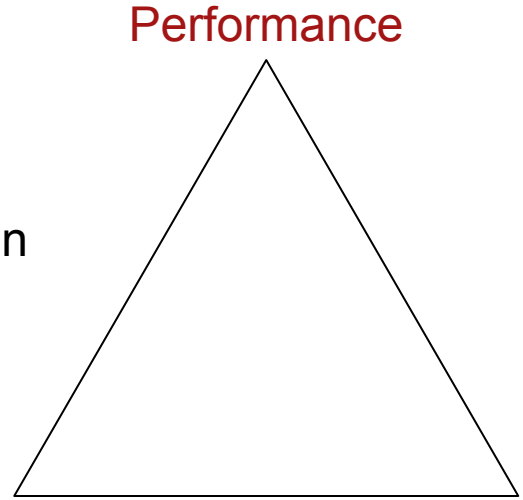
Performance, Portability, Productivity

Write the model in CUDA



Performance, Portability, Productivity

Write the model in python



DSLs in NWP & Climate

Typically aim to capture the numerical patterns in NWP or Climate model

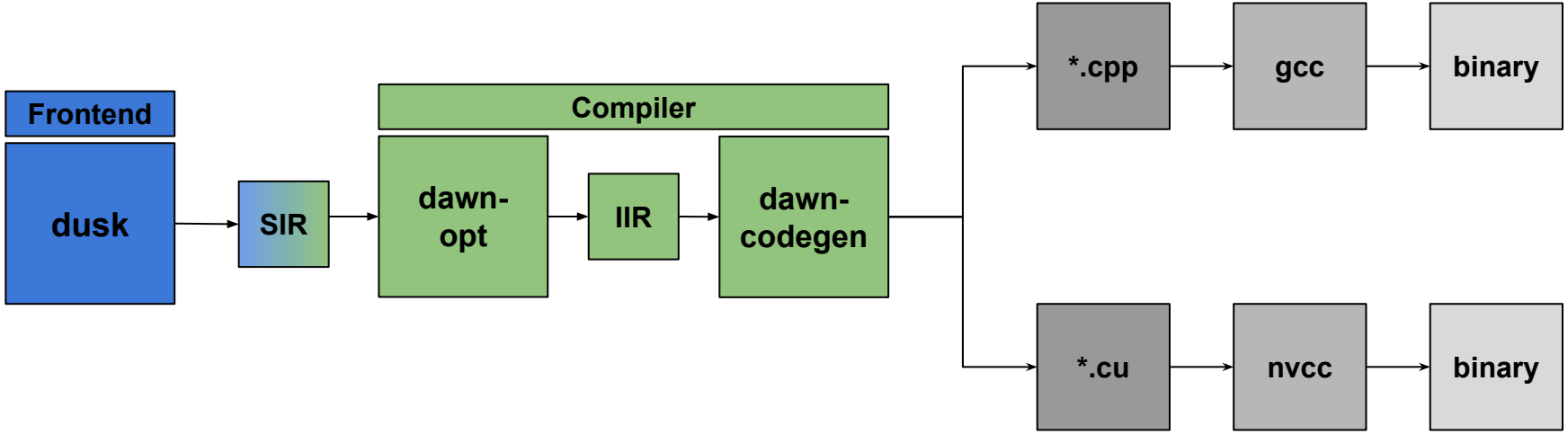
- Finite Differences
  - gtclang + dawn
  - gridtools
  - gt4py
- Finite Volumes
  - dusk + dawn
  - cdsl + dawn
- Combined (Finite Difference / Element / Volume)
  - PsyClone
- ...

DSLs in NWP & Climate

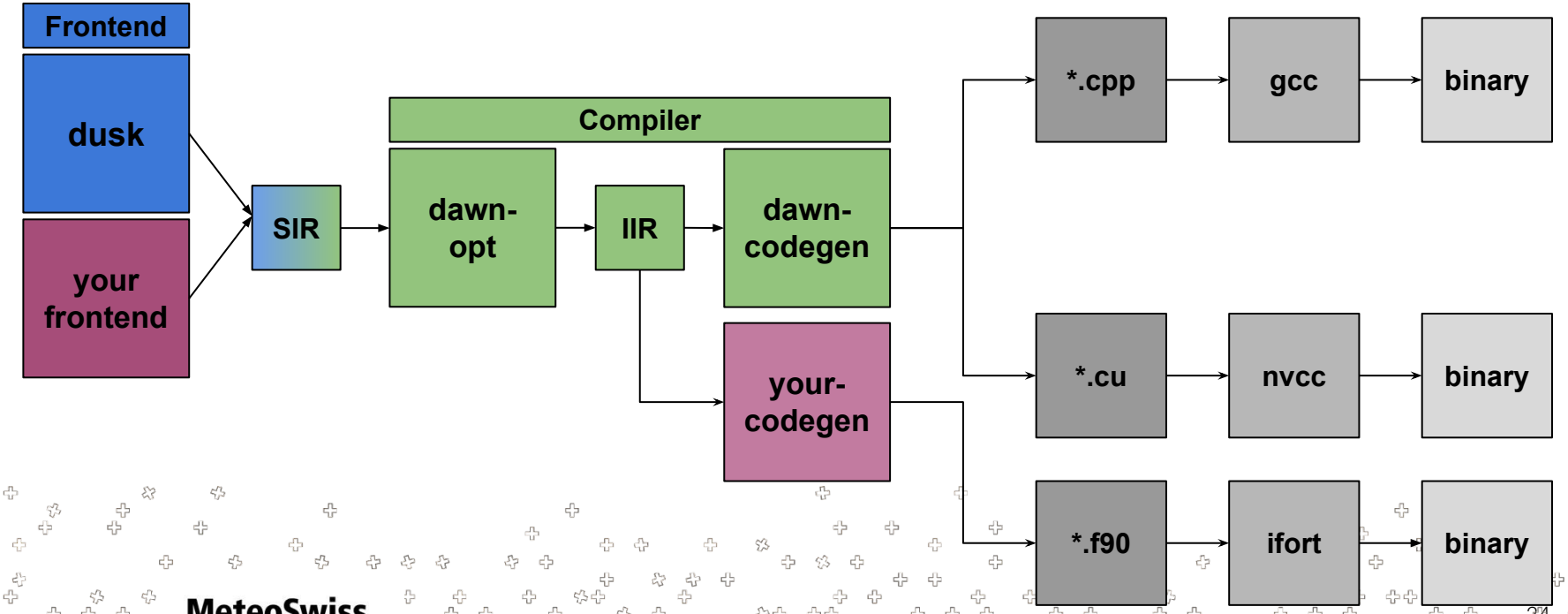
Typically aim to capture the numerical patterns in NWP or Climate model

- Finite Differences
  - gtclang + **dawn**
  - gridtools
  - gt4py
- Finite Volumes
  - **dusk** + **dawn**
  - cdsl + **dawn**
- Combined (Finite Difference / Element / Volume)
  - PsyClone
- ...

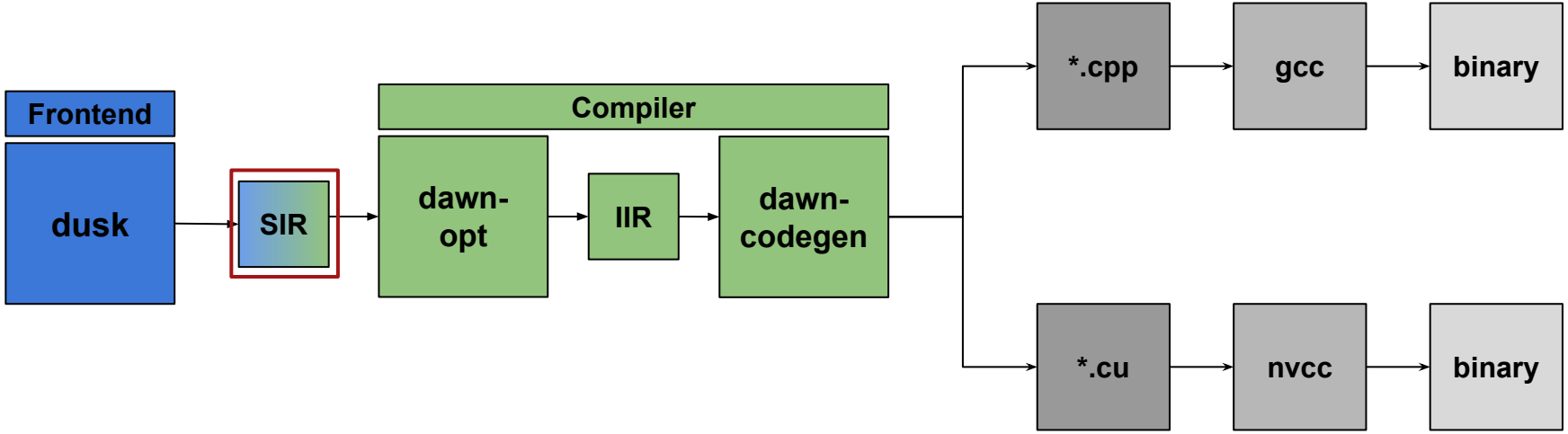
dawn Toolchain - Closer Look



dawn Toolchain - Closer Look



dawn Toolchain - Closer Look



DSL Toolchain - Intermediate Representations

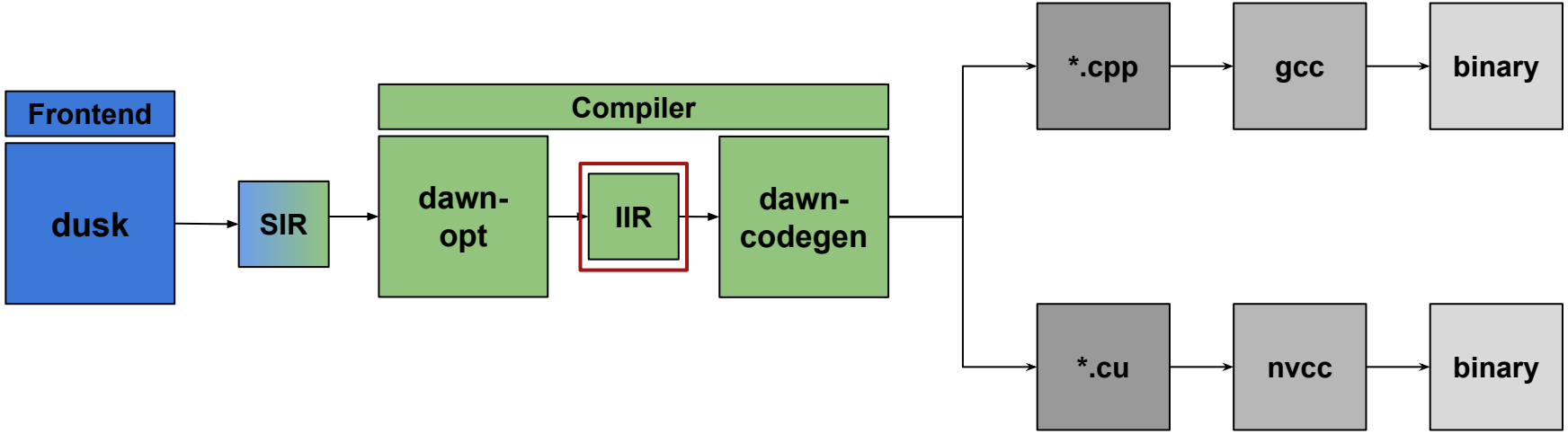
SIR


```
"stencils": [
 {
  "ast": {
   "root": {
    "blockStmt": {
     "statements": [
      {
       "verticalRegionDeclStmt": {
        "verticalRegion": {
         "ast": {
          "root": {
           "blockStmt": {
            "statements": [
             {
              "exprStmt": {
               "expr": {
                "assignmentExpr": {
                 "left": {
                  "fieldAccessExpr": {
                   "name": "rot vec",
                   "unstructuredOffset": {}
                  }
                 },
                 "op": "=",
                 "right": {
                  "reductionOverNeighborExpr": {
                   "op": "+",
                   "rhs": {
                    "binaryOperator": {
                     "left": {
                      "fieldAccessExpr": {
                       "name": "vec",
                       "unstructuredOffset": {
                        "hasOffset": true
```


...

- **A**bstract **S**yntax **T**ree representation close to DSL
- Interface for frontends
- Format well suited for translation
- Not meant to be human-readable

dawn Toolchain - Closer Look



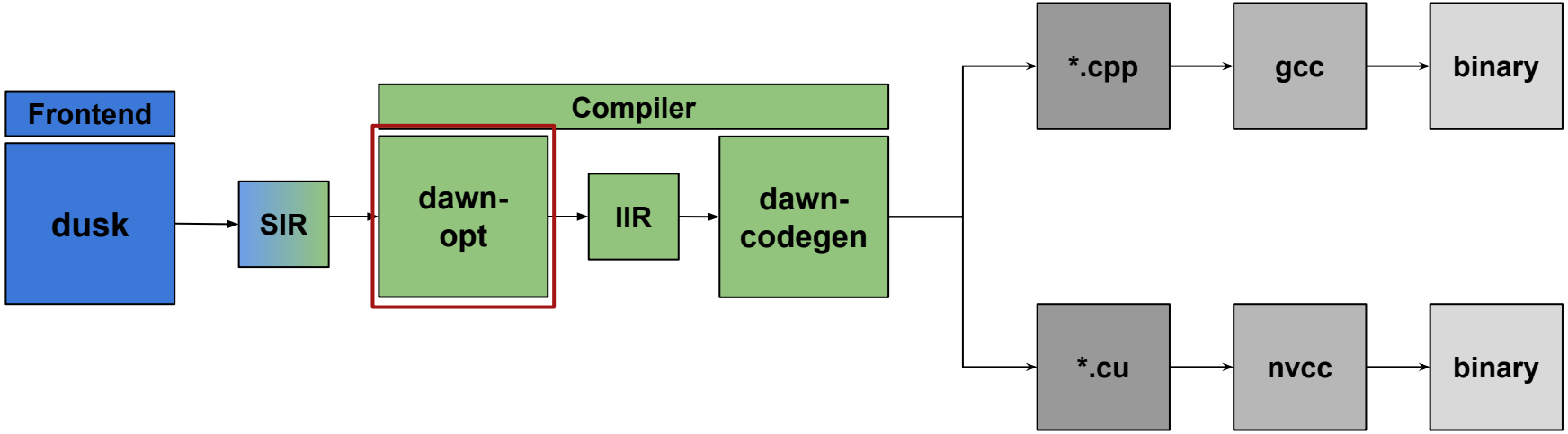
DSL Toolchain - Intermediate Representations

IIR


```
"stencils": [
  {
   "multiStages": [
    {
     "stages": [
      {
       "doMethods": [
        {
         "ast": {
          "block_stmt": {
           "statements": [
            {
             "expr_stmt": {
              "expr": {
               "assignment_expr": {
                "left": {
                 "field_access_expr": {
                  "name": "rot_vec",
                  "vertical_shift": 0,
                  "vertical_indirection": "",
                  "zero_offset": {},
                  "argument_map": [
                   -1,
                   -1,
                   -1
                  ],
                  "argument_offset": [
                   0,
                   0,
                   0
                  ],
                  "negate_offset": false,
...
```


- Also tree-like representation, with ASTs nested inside
- Contains many more details, collected during analyses
- Format well suited for transformations (e.g. through visitors)
- Compiler passes transform valid IIR into valid IIR

dawn Toolchain - Closer Look

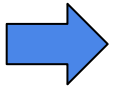


Optimization: one-time stencil inlining

```
@stencil
def my_stencil(
 field_a: Field[Vertex, K],
 field_c: Field[Edge, K]
) -> None:

 field_b: Field[Vertex, K]

 with levels_upward:
   field_b = field_a + 1.0
   field_c = sum_over(Edge > Vertex,
                      field_b)
```



```
__global__ void fused_kernel(double *field_a, double *field_c) {
 ... // e.g. loop over k
   double sum = 0.0;
   for (int nbhIter = 0; nbhIter < E_V_SIZE; nbhIter++) {
     int nbhIdx = evTable[pidx * E_V_SIZE + nbhIter];
     double local_field_b = field_a[k * NumVertices + nbhIdx] +1.0;
     sum += local_field_b;
   }
   field_c[k * NumEdges + pidx] = sum;
 ...
}
```

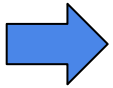
Also when there's an **offset data dependency** we can get rid of accesses to temporary `field_b` and remove its pre-computation.

Optimization: one-time stencil inlining

```
@stencil
def my_stencil(
 field_a: Field[Vertex, K],
 field_c: Field[Edge, K]
) -> None:

field_b: Field[Cell, K]

with levels_upward:
  field_b = sum_over(Cell > Vertex,
                     field_a)
  field_c = sum_over(Edge > Cell,
                     field_b)
```

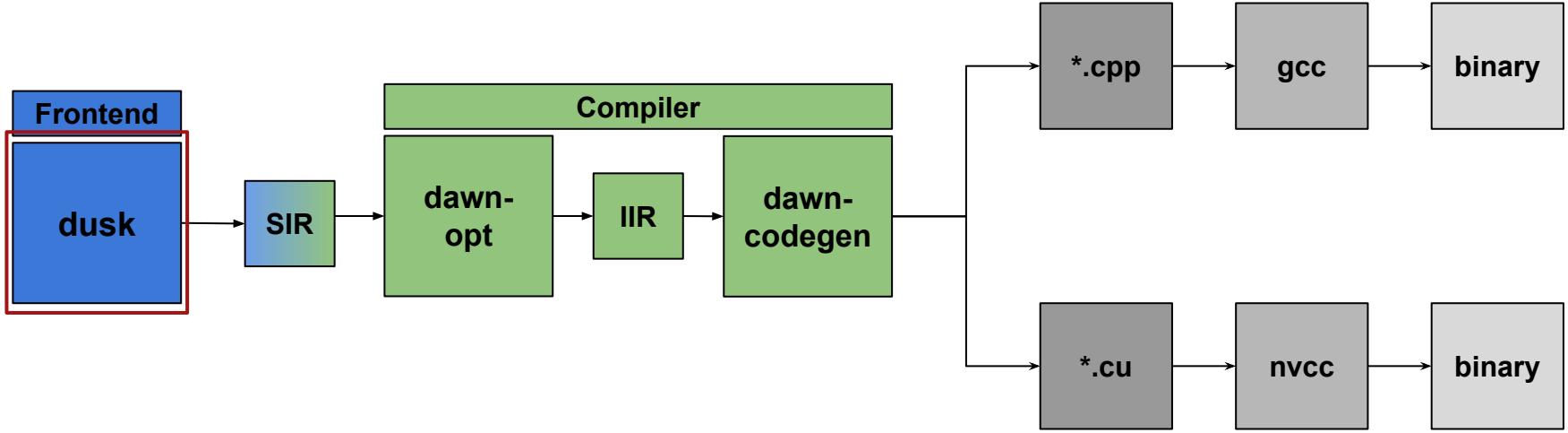


```
__global__ void fused_kernel(double *field_a, double *field_c) {
 ... // e.g. loop over k
double sum_0 = 0.0;
for (int nbhIter_0 = 0; nbhIter_0 < E_C_SIZE; nbhIter_0++) {
  int nbhIdx_0 = ecTable[pidx * E_C_SIZE +nbhIter_0];
  double sum_1 = 0.0;
  for (int nbhIter_1 = 0; nbhIter_1 < C_V_SIZE; nbhIter_1++) {
    int nbhIdx_1 = cvTable[nbhIdx_0 * C_V_SIZE + nbhIter_1];
    sum_1 += field_a[k * NumVertices +nbhIdx_1];
  }
  double local_field_b = sum_1;
  sum_0 += local_field_b;
}
field_c[k * NumEdges + pidx] =sum_0;
 ...
}
```

Same optimization. Now first statement is a reduction.

Need to nest reduction loops in the generated code.

dawn Toolchain - Closer Look



Designing a DSL for FVM (ICON)

- vertical vorticity

$$\zeta_q = \frac{1}{\hat{a}_q} \sum_j^{j \in \mathcal{E}(q)} v_j^n \cdot \hat{l}_j \cdot f_j \qquad | \forall_q \in \mathcal{V}$$

- divergence of an edge flux Φ

$$\nabla(\phi)_q = \frac{1}{a_q} \sum_j^{j \in \mathcal{E}(q)} \phi_j \cdot l_j \cdot f_j \qquad | \forall_q \in \mathcal{C}$$

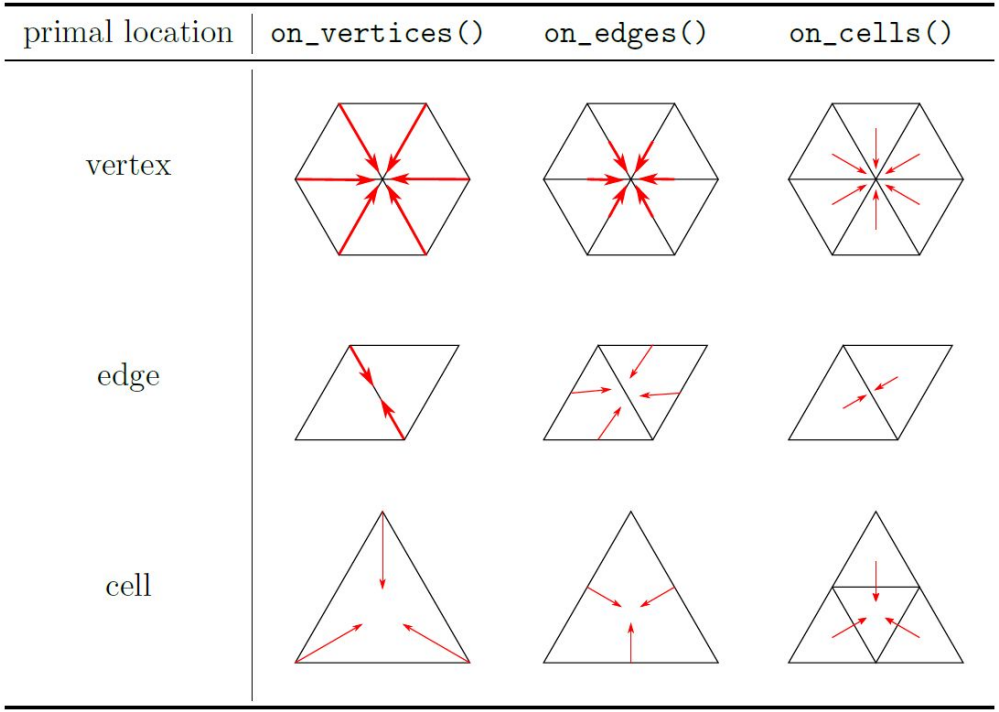
The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core - Zängl et al

Designing a DSL for FVM (ICON)

Fundamental concept: **reduction**

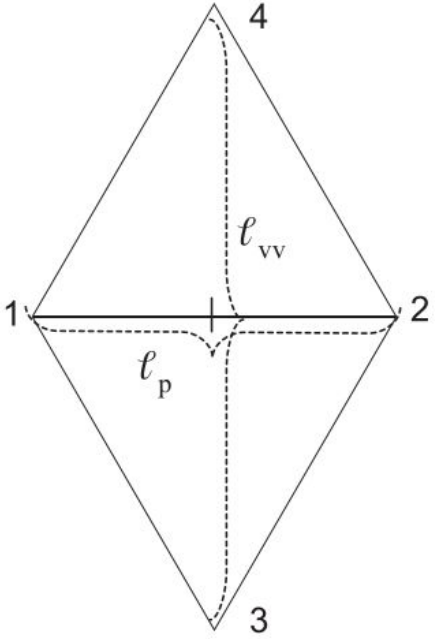
Early idea:

```
cellField div_c;
edgeField flux, n;
div_c = reduce(flux*n)
```



Designing a DSL for FVM (ICON)

There are more general computations required than first order FVM stencils



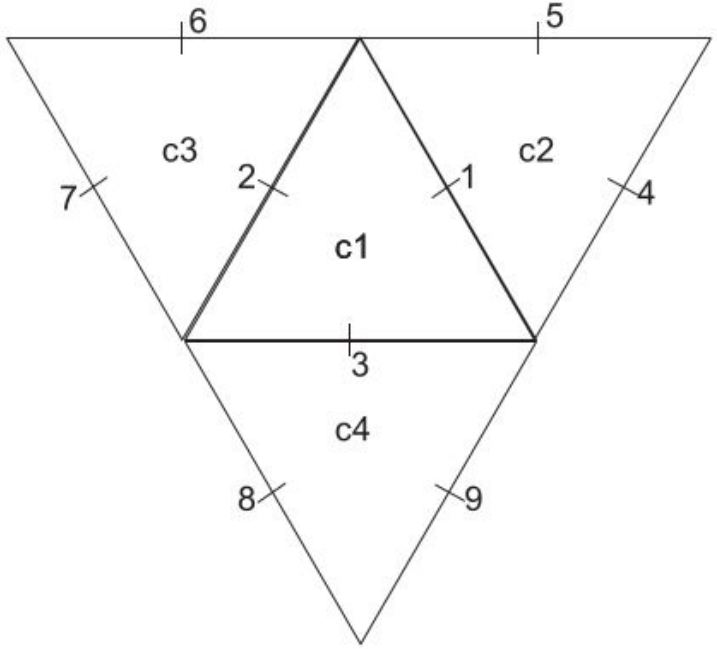
Obtain Laplacian via FD

$$\nabla^2(v_n) = \frac{v_{n2} + v_{n1} - 2v_n}{l_p^2} + \frac{v_{n4} + v_{n3} - 2v_n}{l_{vv}^2}$$

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

Designing a DSL for FVM (ICON)

There are more general computations required than first order FVM stencils



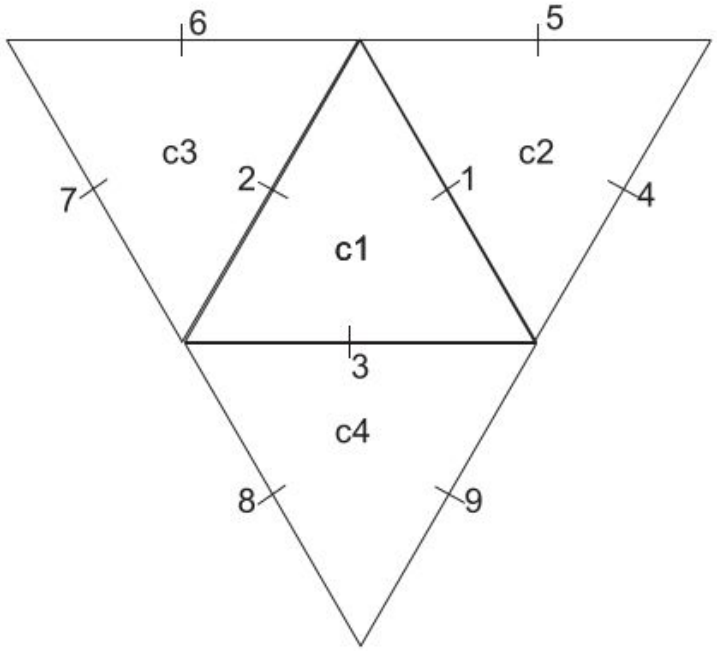
High order interpolation (velocity & divergence averaging)

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

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

Designing a DSL for FVM (ICON)

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



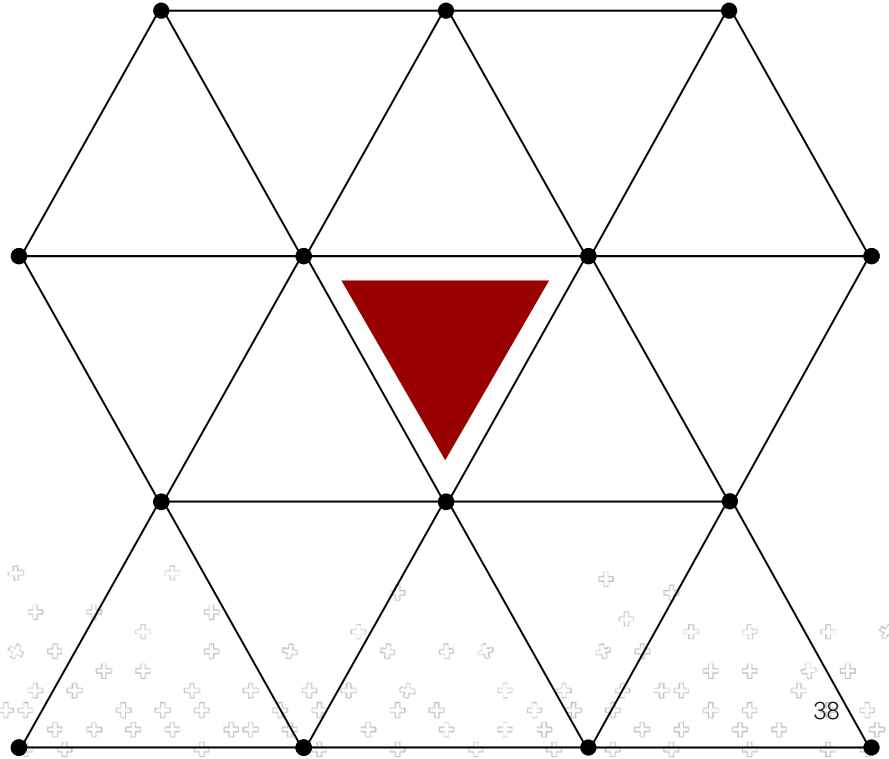
$$f(c_1) = \sum_{j=1}^{9} f_j w_j$$

```
@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

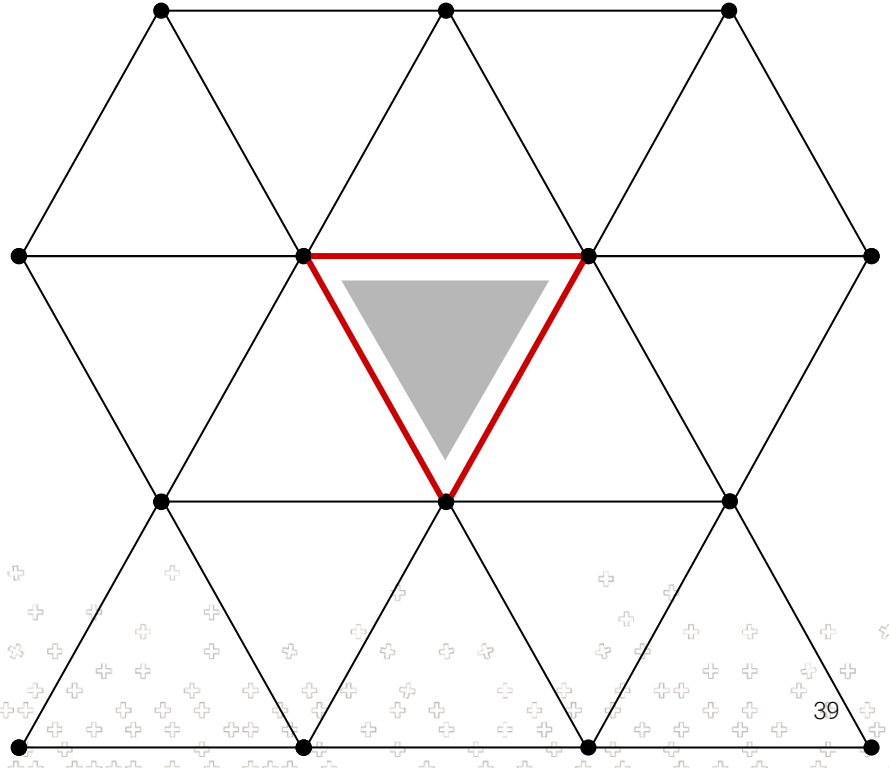
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



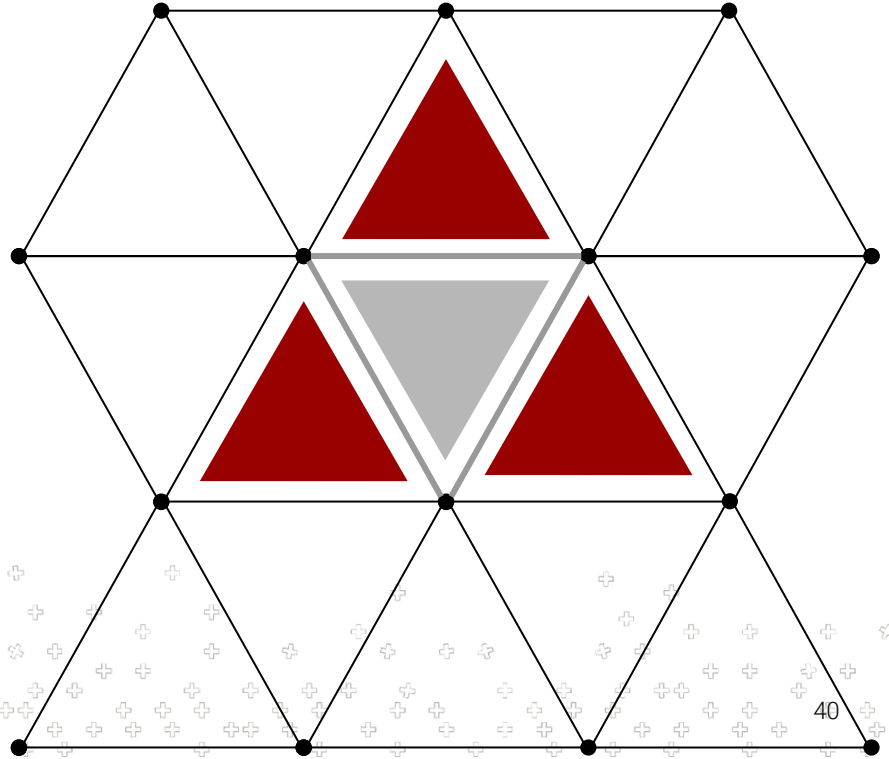
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



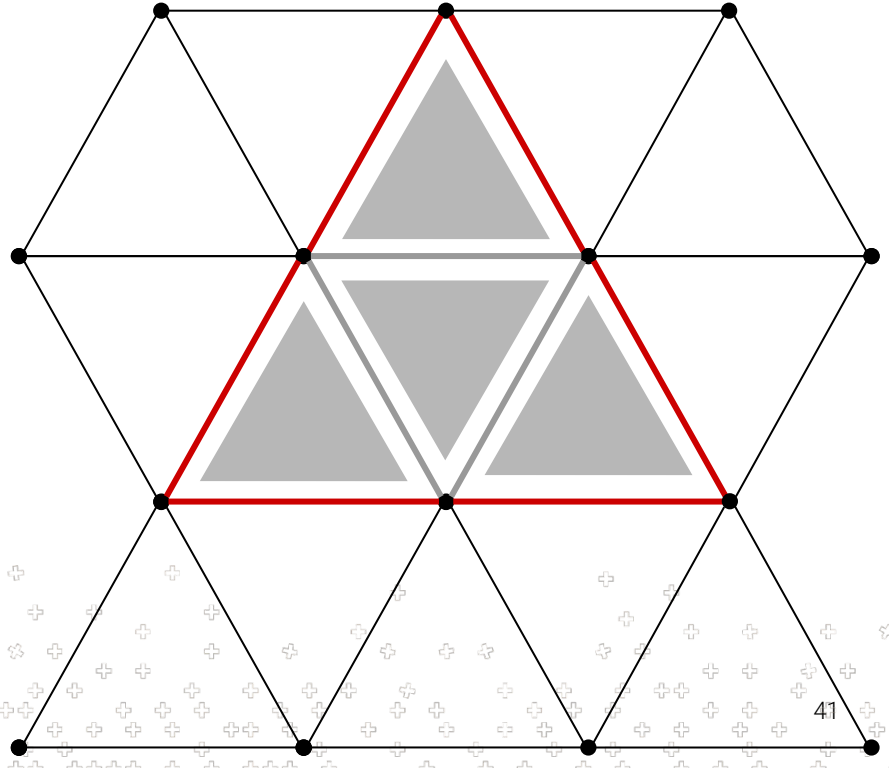
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



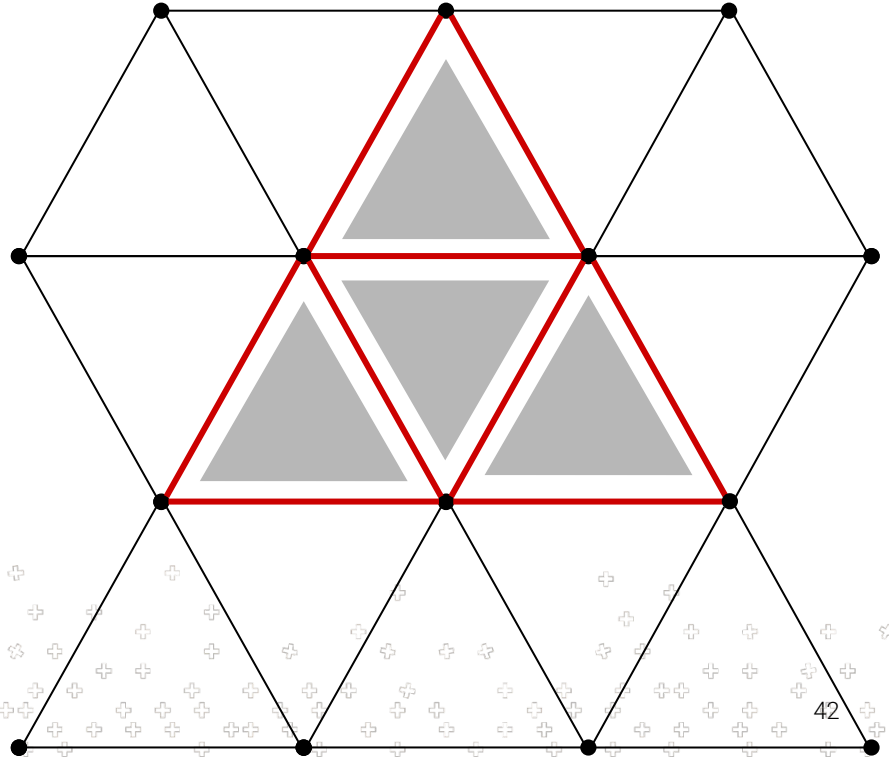
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



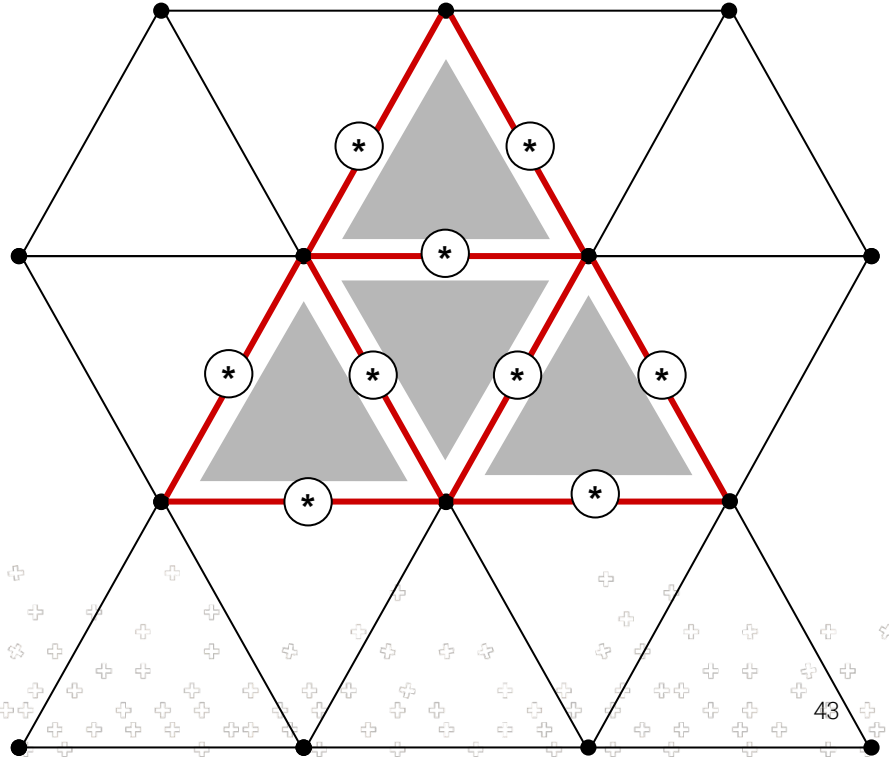
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



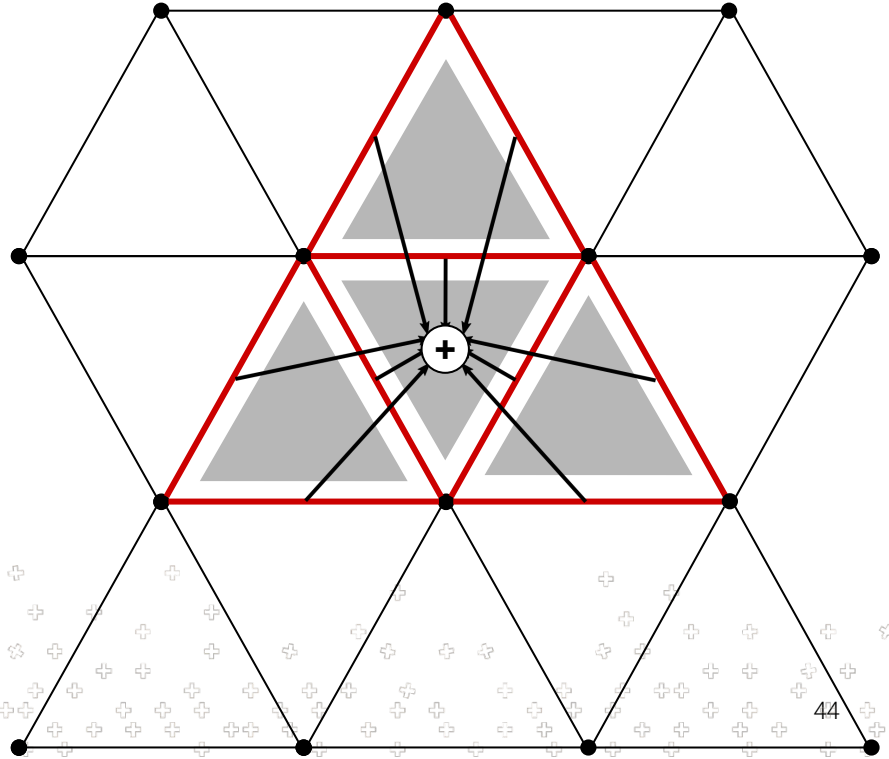
Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



Neighbor Chains - Case Study - RBF Interpolation

```
@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)
```



Summary: DSL for NWP - Addressing the "3 Ps"

Portability

- If a new architecture is required, a new backend needs to be written
  - No duplication of model code
  - Usually (way) less effort

Productivity

- Language tailored to problem space

Performance

- Optimization passes
- No best practice requirements for generated code
  - usually slow but readable debug backend + different fast but less readable backends

Summary: DSL for NWP - Addressing the "3 Ps"

Portability

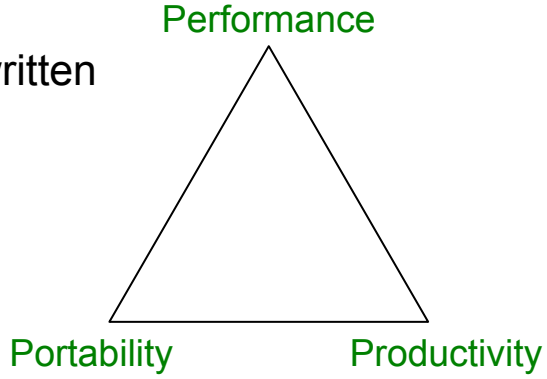
- If a new architecture is required, a new backend needs to be written
  - No duplication of model code
  - Usually (way) less effort

Productivity

- Language tailored to problem space

Performance

- Optimization passes
- No best practice requirements for generated code
  - usually slow but readable debug backend + different fast but less readable backends



Q&A

Questions?