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 dusk

A minimal and lightweight front-end for dawn

Dusk Overview

Dusk is:

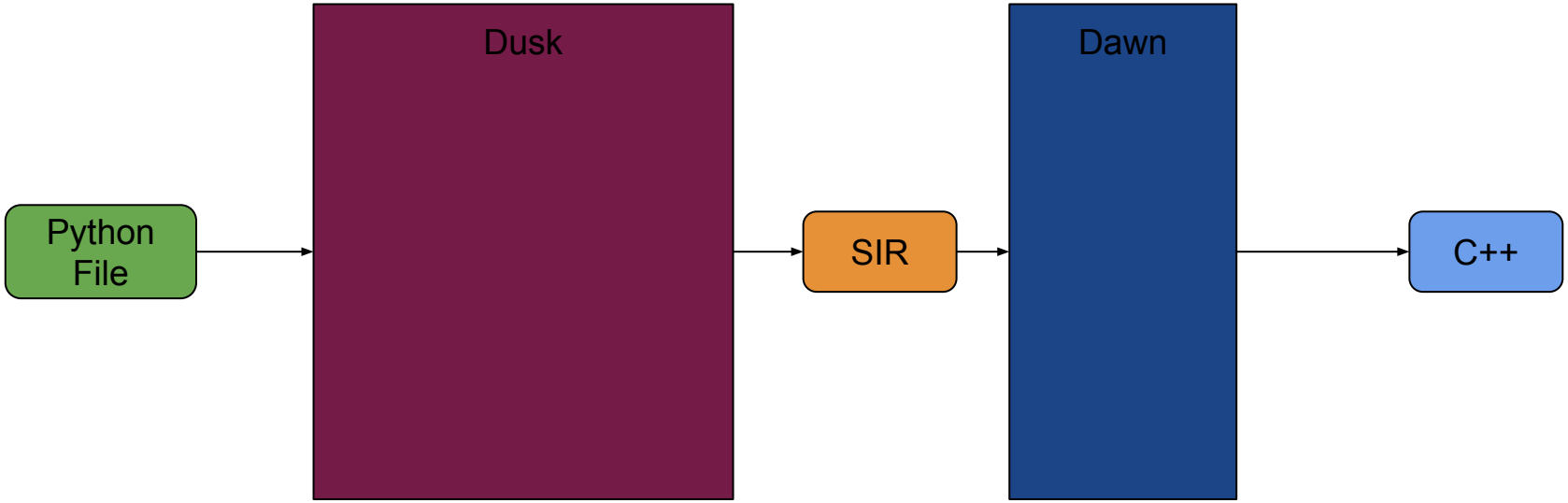
- A programming language
  - *dusk script/dusk stencil language*
  - A domain specific language embedded in Python (Python eDSL)
- A compiler/transpiler
  - Compiles/transpiles *dusk* to SIR
  - Can call into dawn to directly generate C++
- A software component to translate *dusk* to SIR or C++
  - Also contains a Python API to programmatically translate *dusk*
  - Written in Python
  - no dependencies other than Python's standard library

Dusk Overview

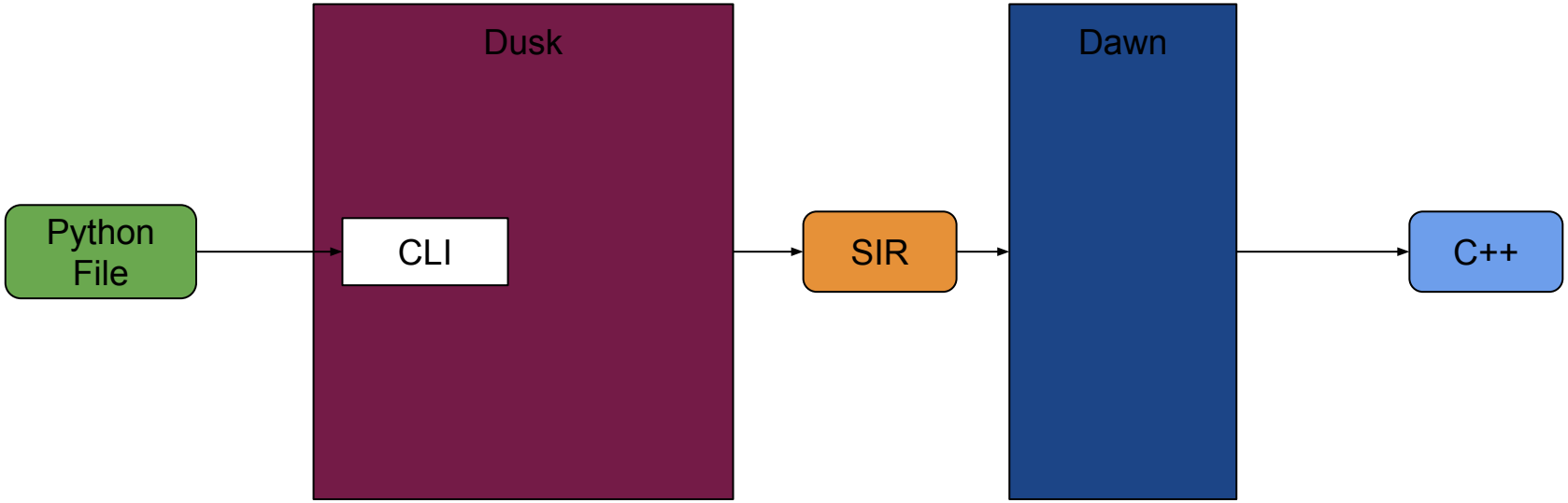
Dusk is:

- A programming language
  - dusk script/dusk stencil language
  - A domain specific language embedded in Python (Python eDSL)
- A compiler/transpiler
  - Compiles/transpiles *dusk* to SIR
  - Can call into dawn to directly generate C++
- A software component to translate *dusk* to SIR or C++
  - Also contains a Python API to programmatically translate *dusk*
  - Written in Python
  - no dependencies other than Python's standard library

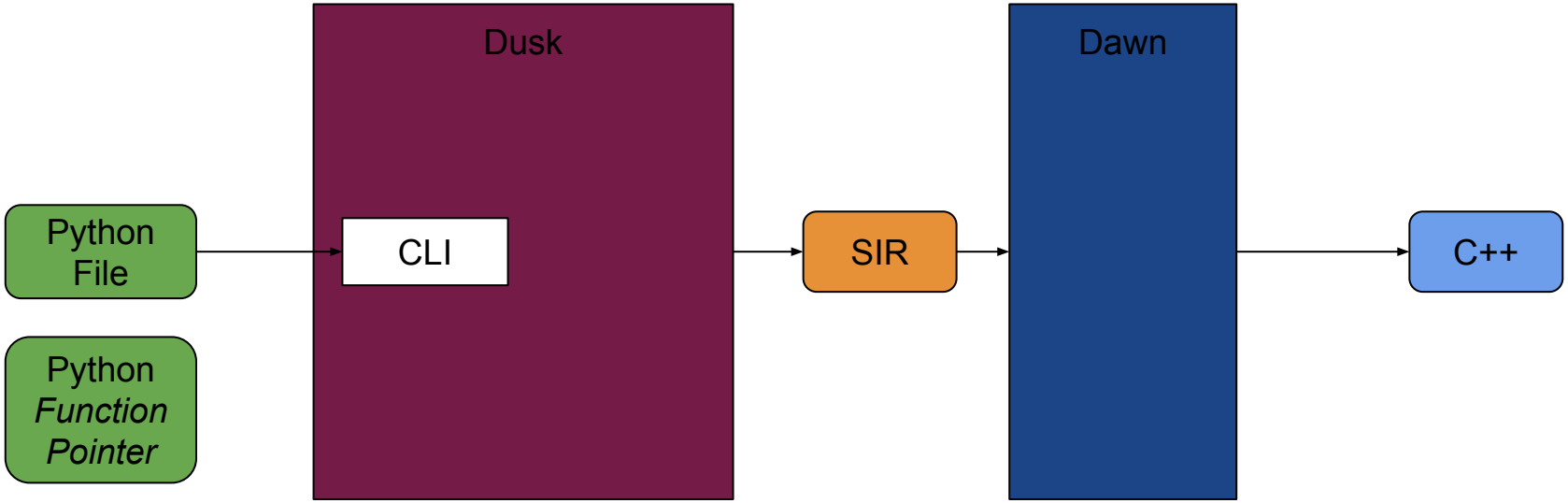
Dusk Framework



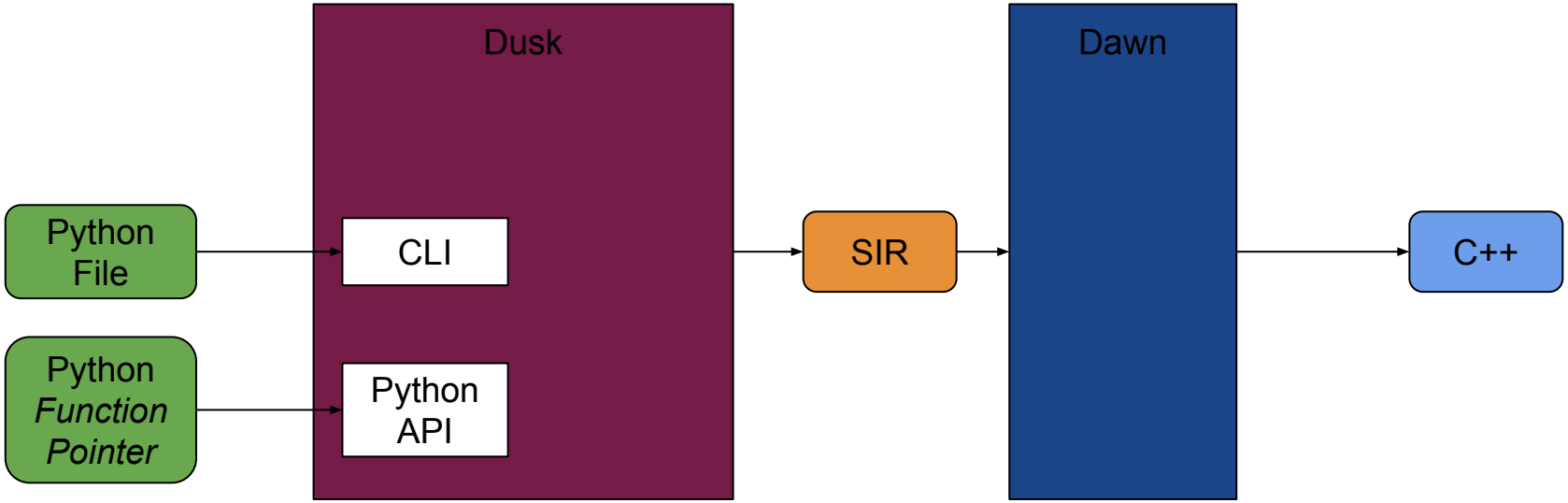
Dusk Framework



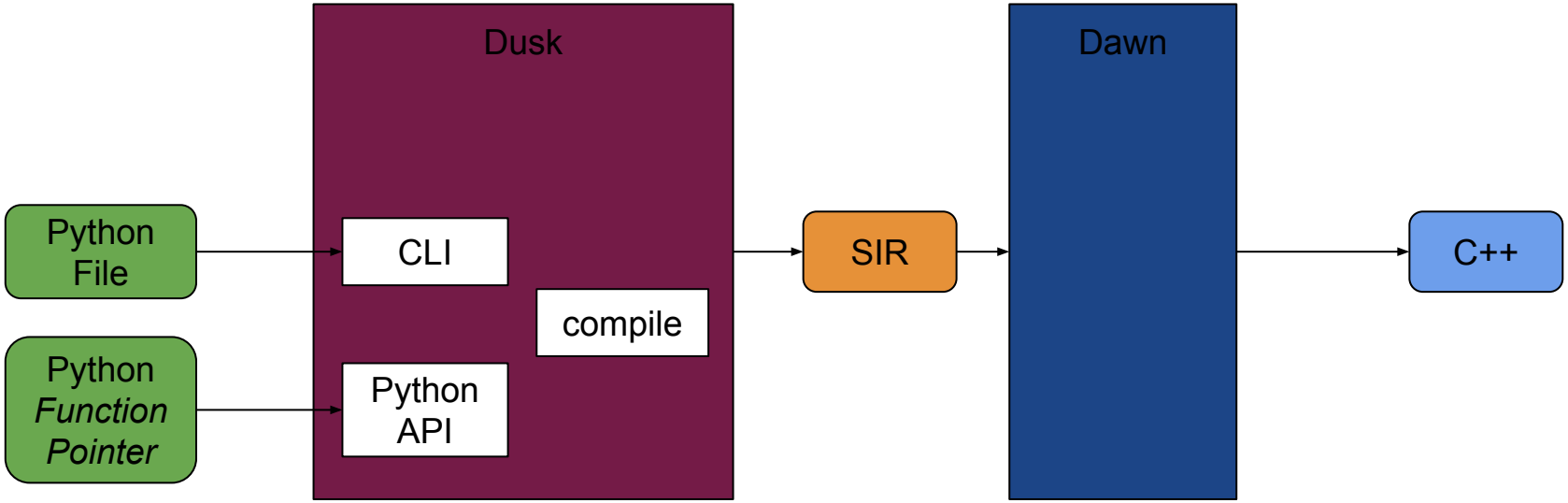
Dusk Framework



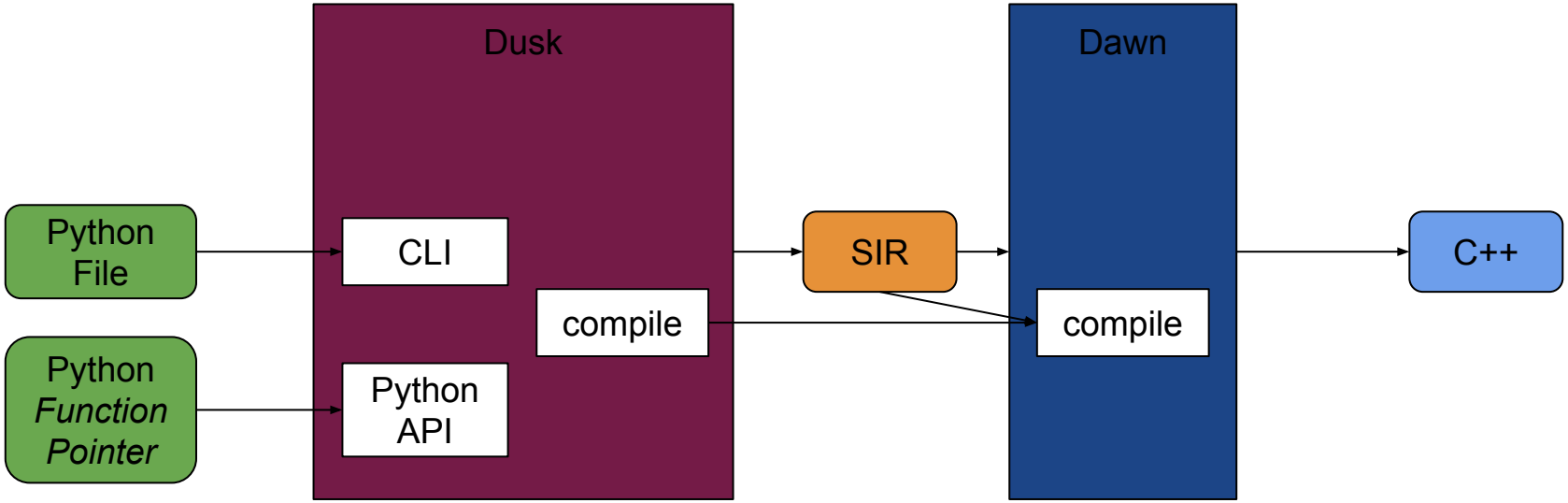
Dusk Framework



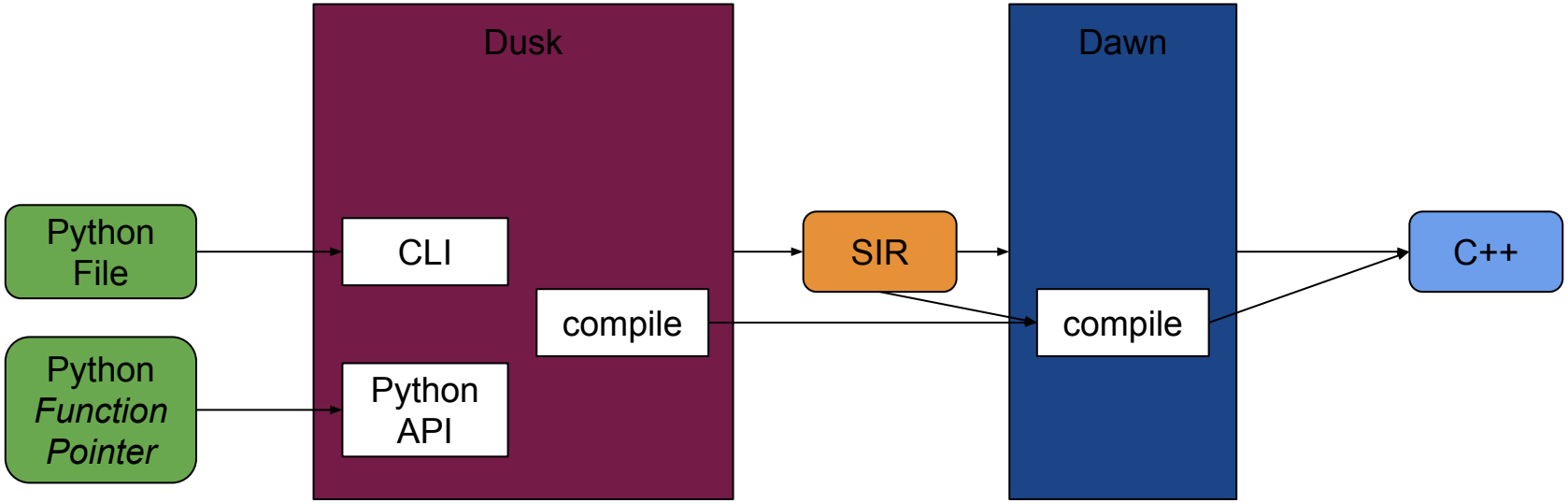
Dusk Framework



Dusk Framework



Dusk Framework



Dusk Overview

Dusk is:

- A programming language
  - *dusk script/dusk stencil language*
  - A domain specific language embedded in Python (Python eDSL)
- A compiler/transpiler
  - Compiles/transpiles dusk to SIR
  - Can call into dawn to directly generate C++
- A software component to translate dusk to SIR or C++
  - Also contains a Python API to programmatically translate dusk
  - Written in Python
  - no dependencies other than Python's standard library

Dusk Overview

Dusk is:

- A programming language
  - *dusk script/dusk stencil language*
  - A domain specific language embedded in Python (Python eDSL)
- A compiler/transpiler
  - Compiles/transpiles *dusk* to SIR
  - Can call into
- A software compo
  - Also contains a Python API to programmatically translate *dusk*
  - Written in Python
  - no dependencies other than Python's standard library

How does dusk fit into weather & climate models?
Where are the stencils in a climate or weather model?

Dusk in weather & climate models

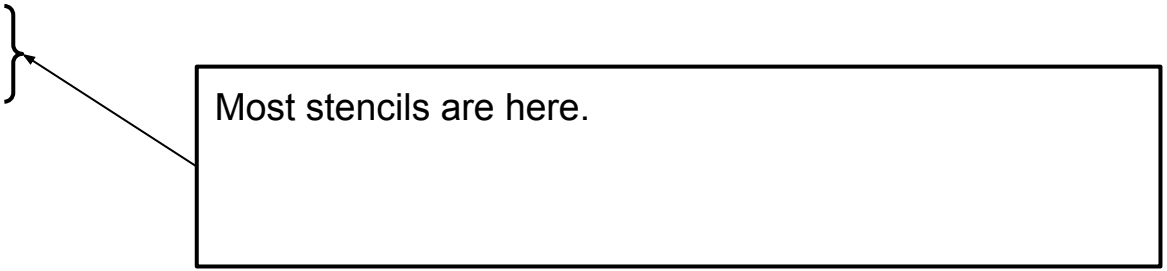
Codebase of a weather or climate model:

- Initialization (MPI, Mesh, etc)
- File Input/Output (data, config, logs, etc)
- Simulation/Calculations
  - Dynamics
  - Physics
  - …
- ...

Dusk in weather & climate models

Codebase of a weather or climate model:

- Initialization (MPI, Mesh, etc)
- File Input/Output (data, config, logs, etc)
- Simulation/Calculations
  - Dynamics
  - Physics
  - …
- ...



Dusk in weather & climate models

Codebase of a weather or climate model:

- Initialization (MPI, Mesh, etc)
- File Input/Output (data, config, logs, etc)
- Simulation/Calculations
  - Dynamics
  - Physics
  - …
- ...

Most stencils are here.
This is a major part of the runtime.

Dusk in weather & climate models

Codebase of a weather or climate model:

- Initialization (MPI, Mesh, etc)
- File Input/Output (data, config, logs, etc)
- Simulation/Calculations
  - Dynamics
  - Physics
  - …
- ...

Most stencils are here.
This is a major part of the runtime.
Let's use hardware accelerators to speed this up.

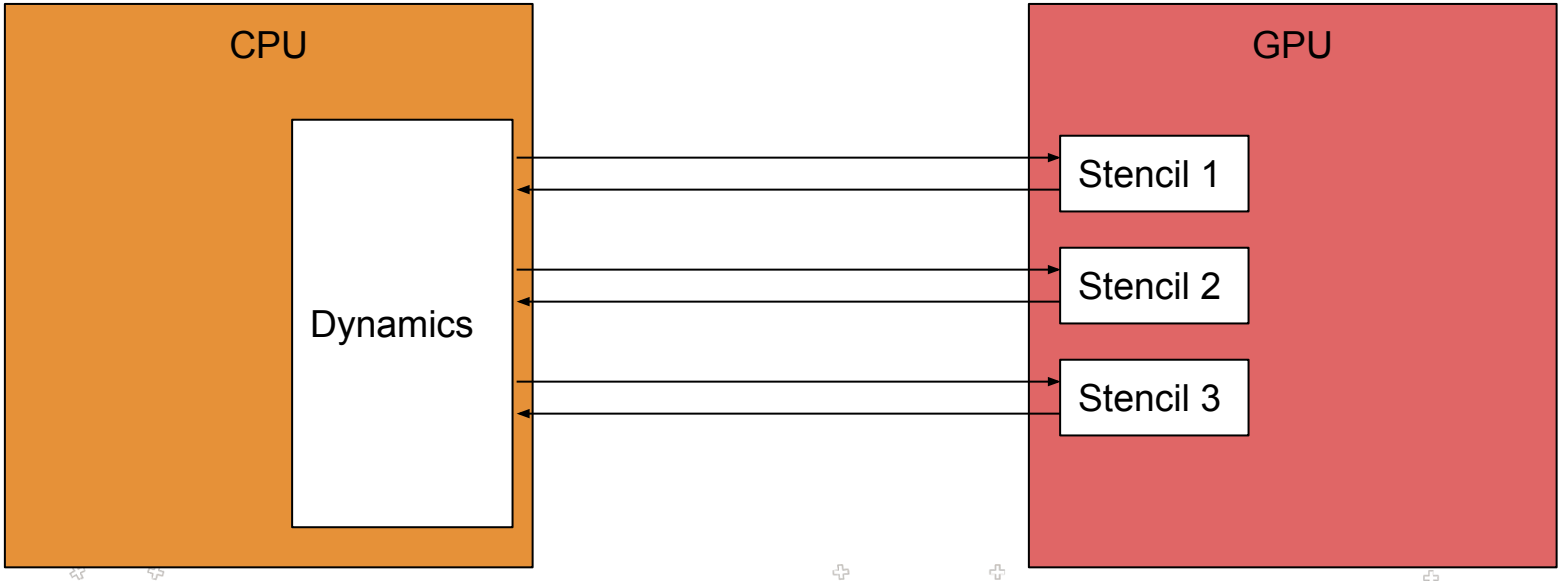
Dusk in weather & climate models

Codebase of a weather or climate model:

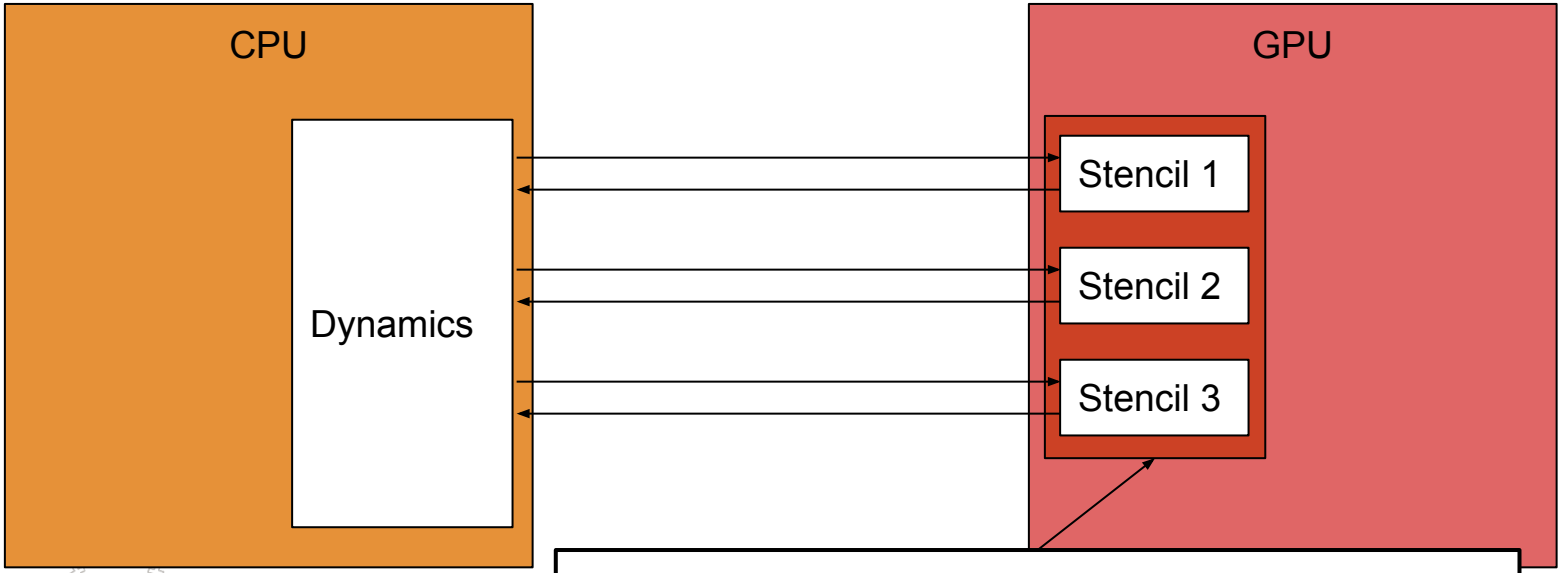
- Initialization (MPI, Mesh, etc)
- File Input/Output (data, config, logs, etc)
- Simulation/Calculations
  - Dynamics
  - Physics
  - …
- ...

Most stencils are here.
This is a major part of the runtime.
Let's use hardware accelerators to speed this up.
E.g., GPUs...

Dusk in weather & climate models

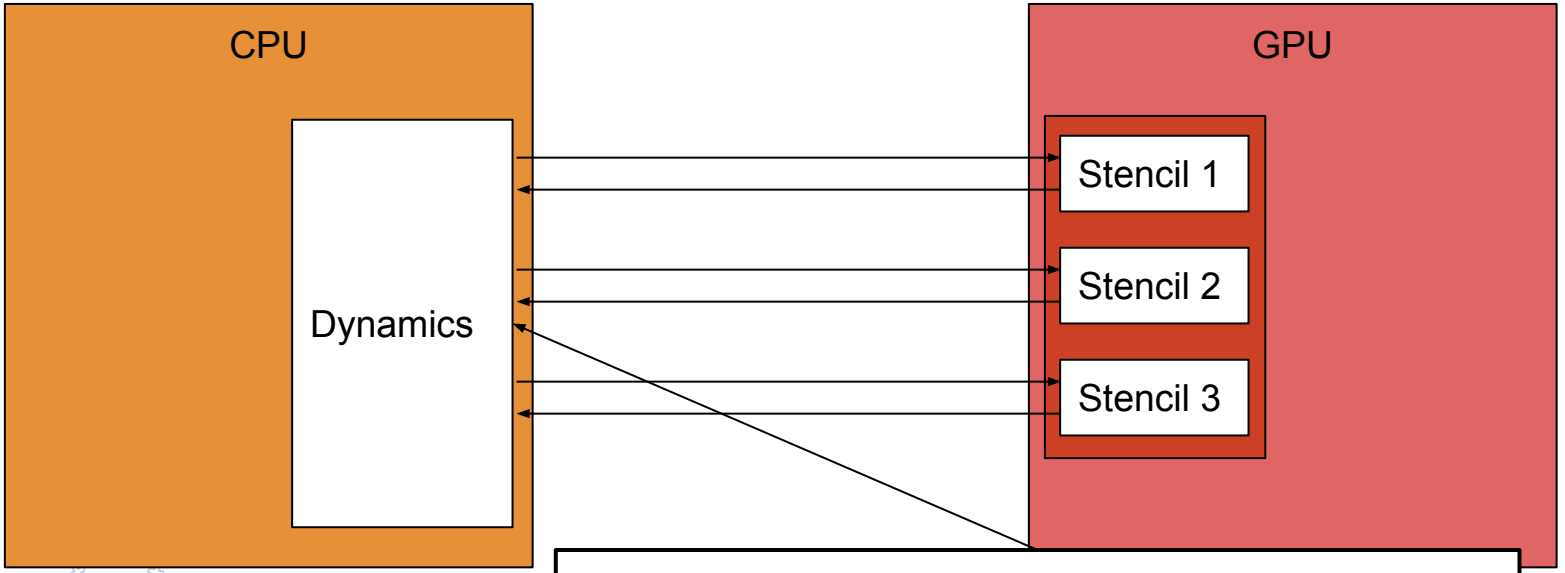


Dusk in weather & climate models

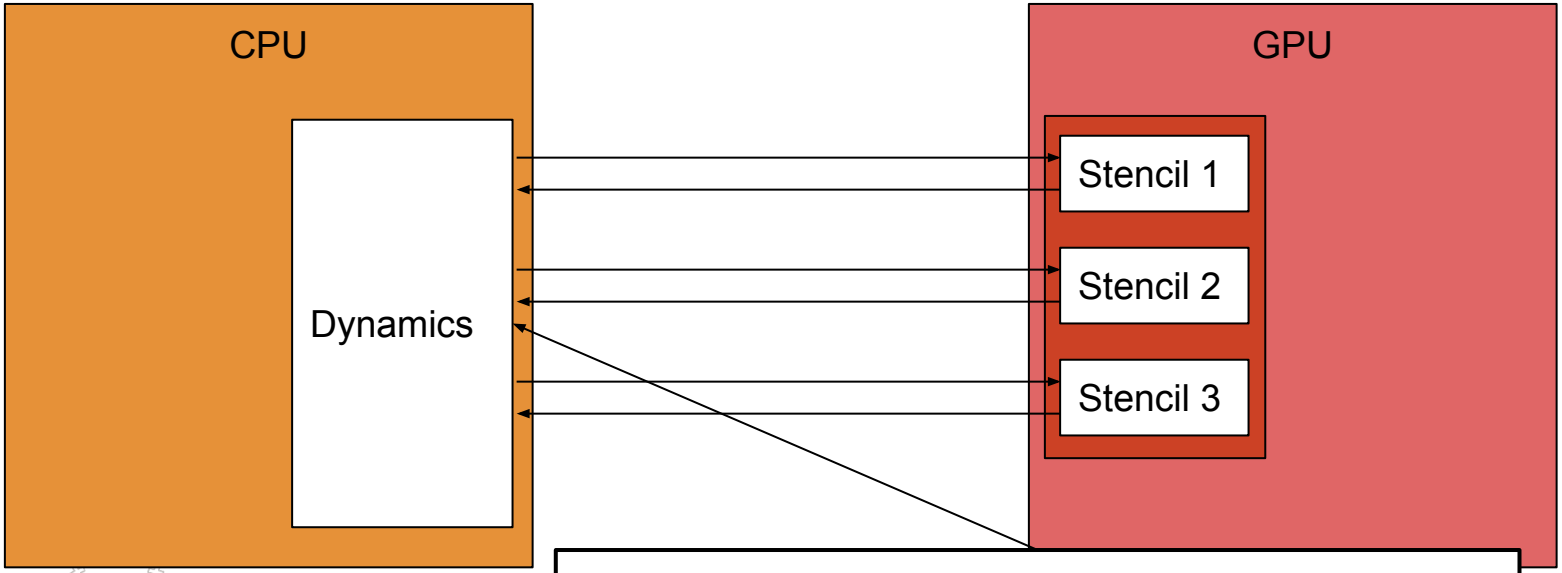


The stencils run on the hardware accelerator.
E.g., a GPU

Dusk in weather & climate models

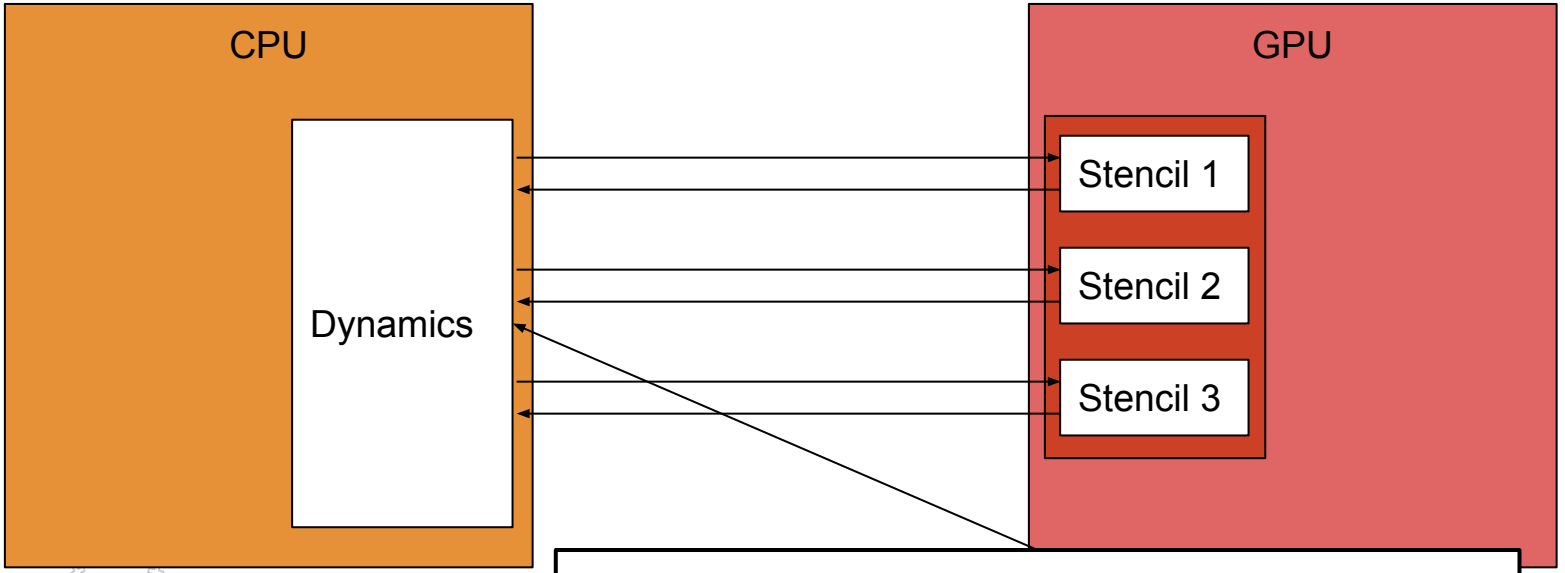


Dusk in weather & climate models



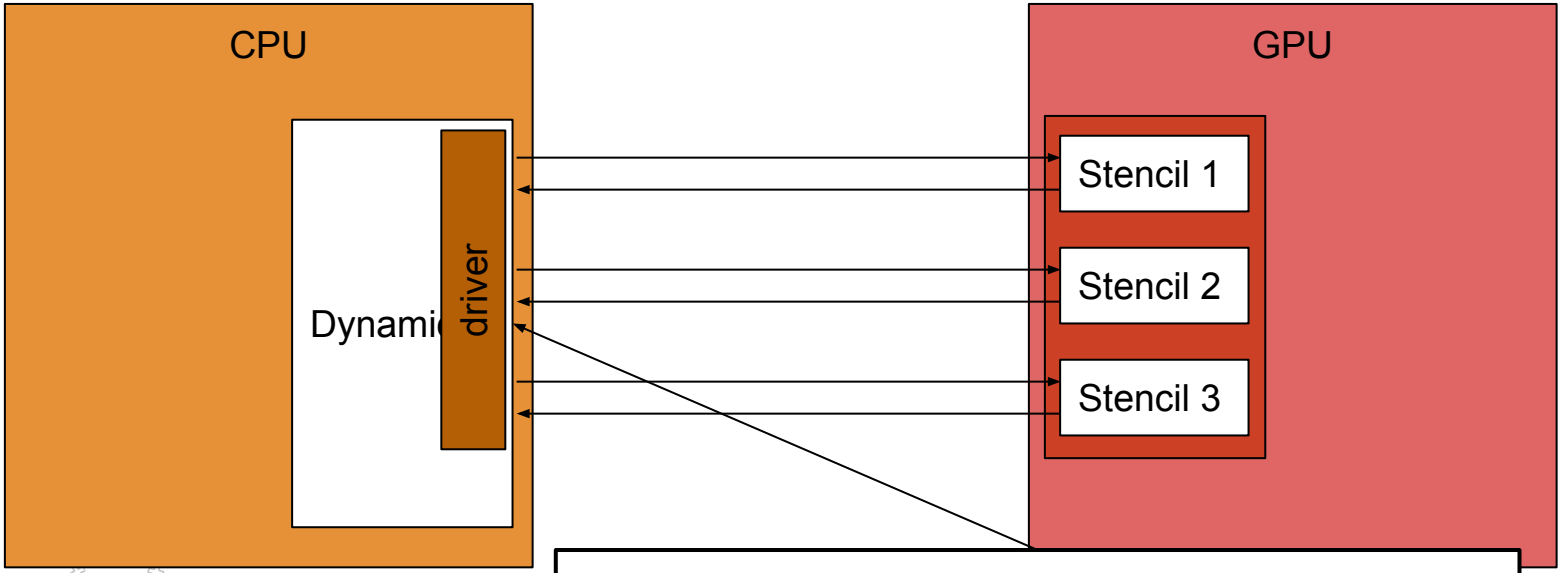
There's still code running on the CPU.
This code decides when and how the stencils are run.

Dusk in weather & climate models



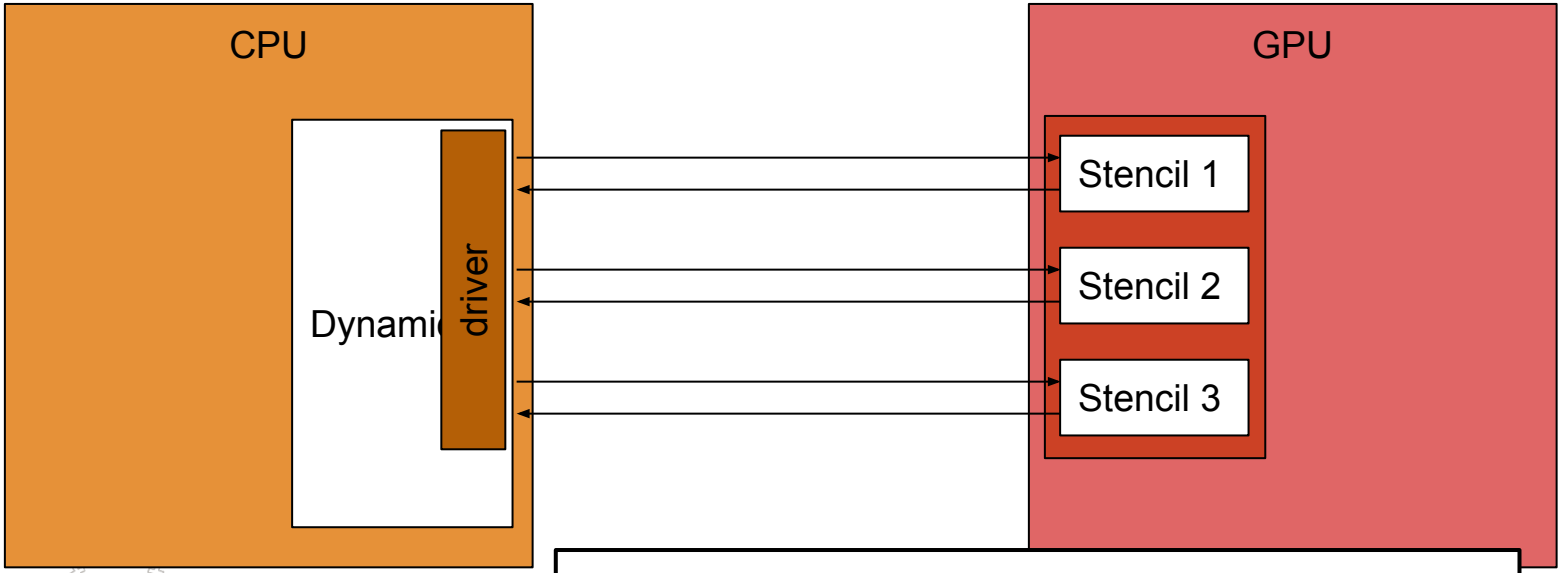
There's still code running on the CPU.
This code decides when and how the stencils are run.
≈ Dynamics component without the stencils

Dusk in weather & climate models



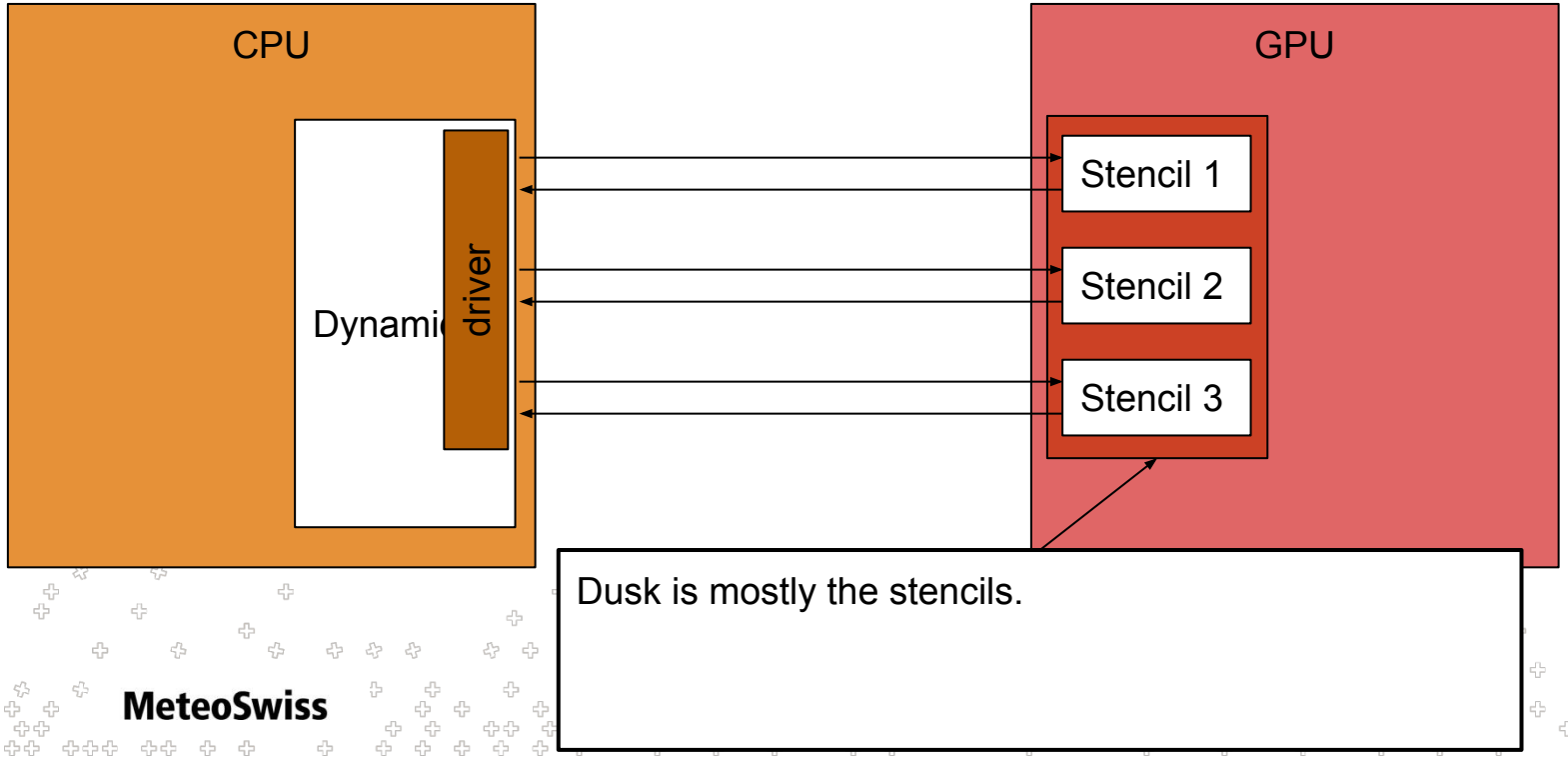
We call this the *driver code*…
It *drives* the stencils (unrelated to hardware drivers).

Dusk in weather & climate models

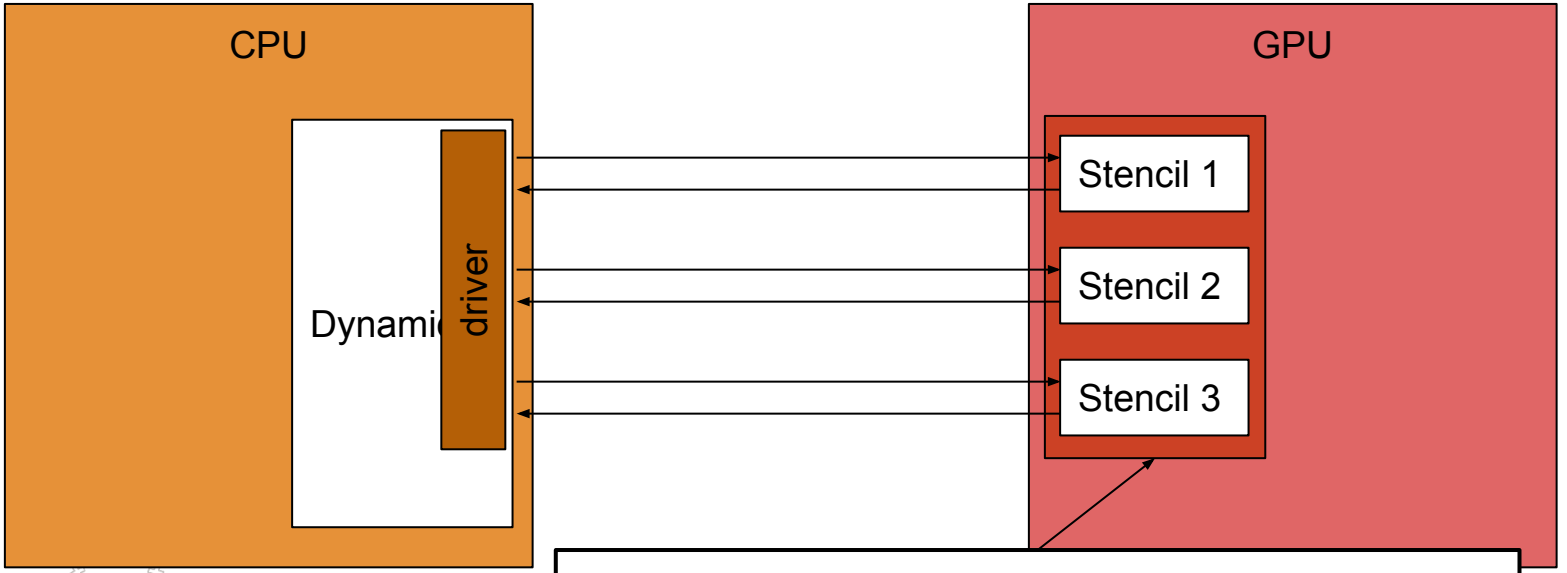


So where is dusk?

Dusk in weather & climate models

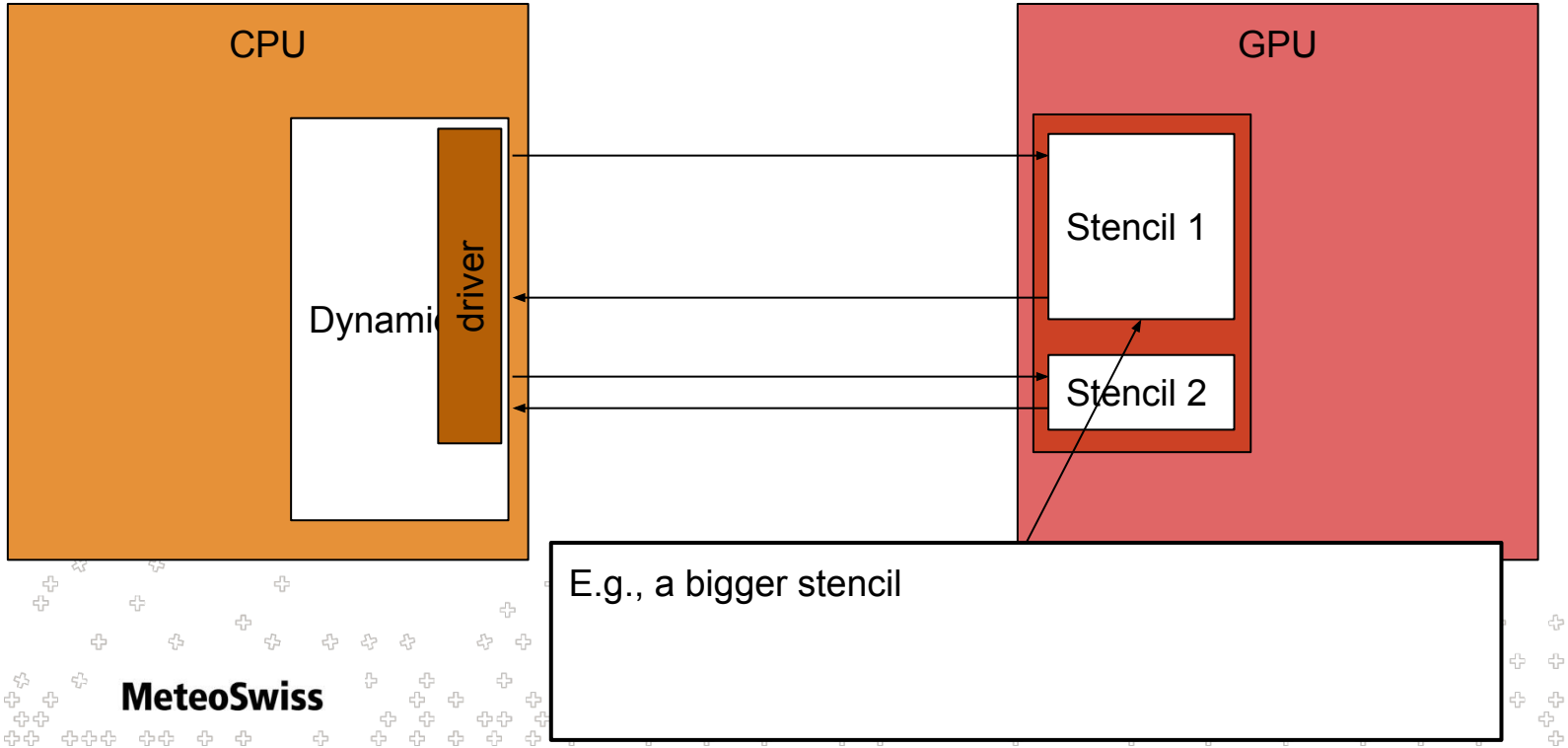


Dusk in weather & climate models

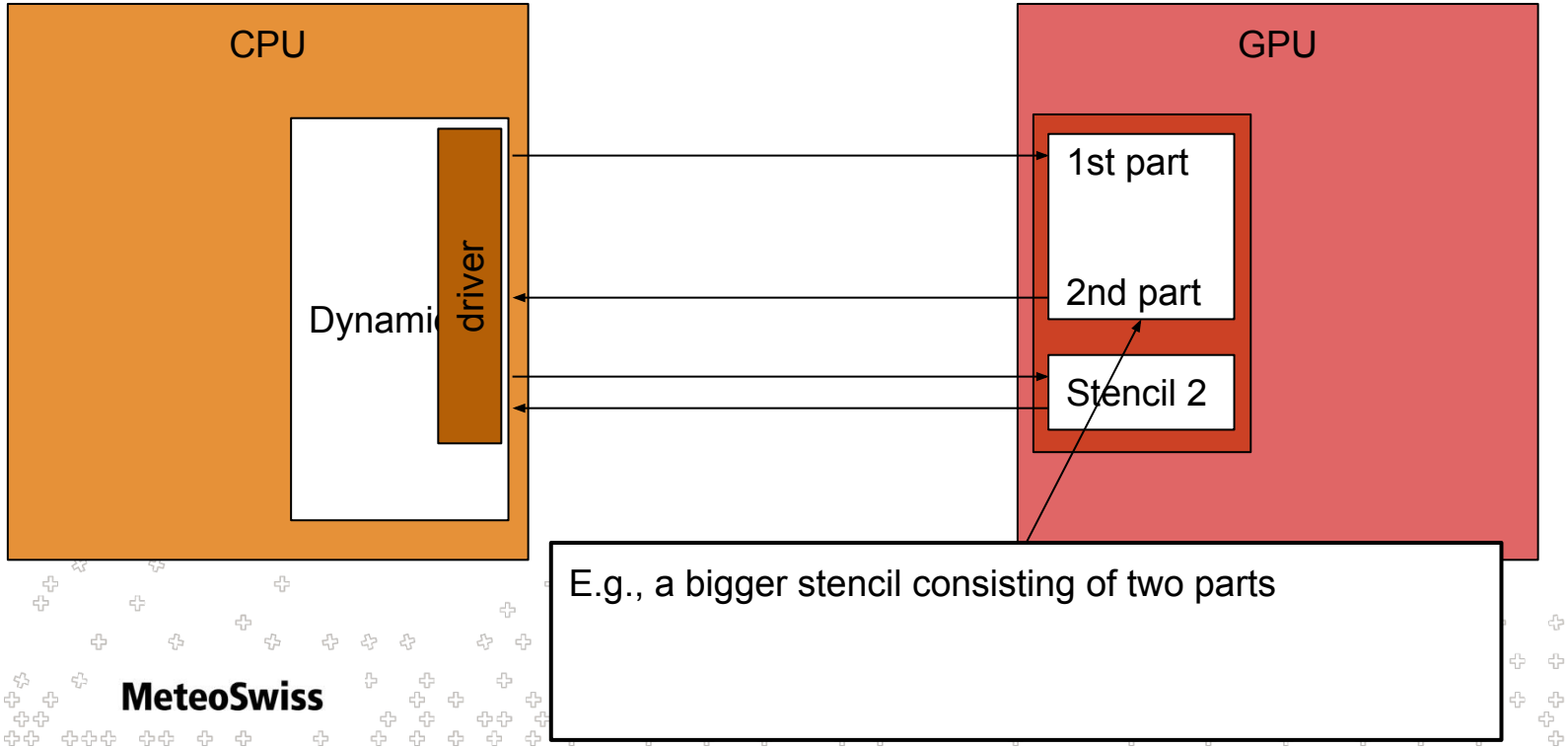


Dusk is mostly the stencils.
However, the line isn't always that clear.
Sometimes it can be quite blurry.

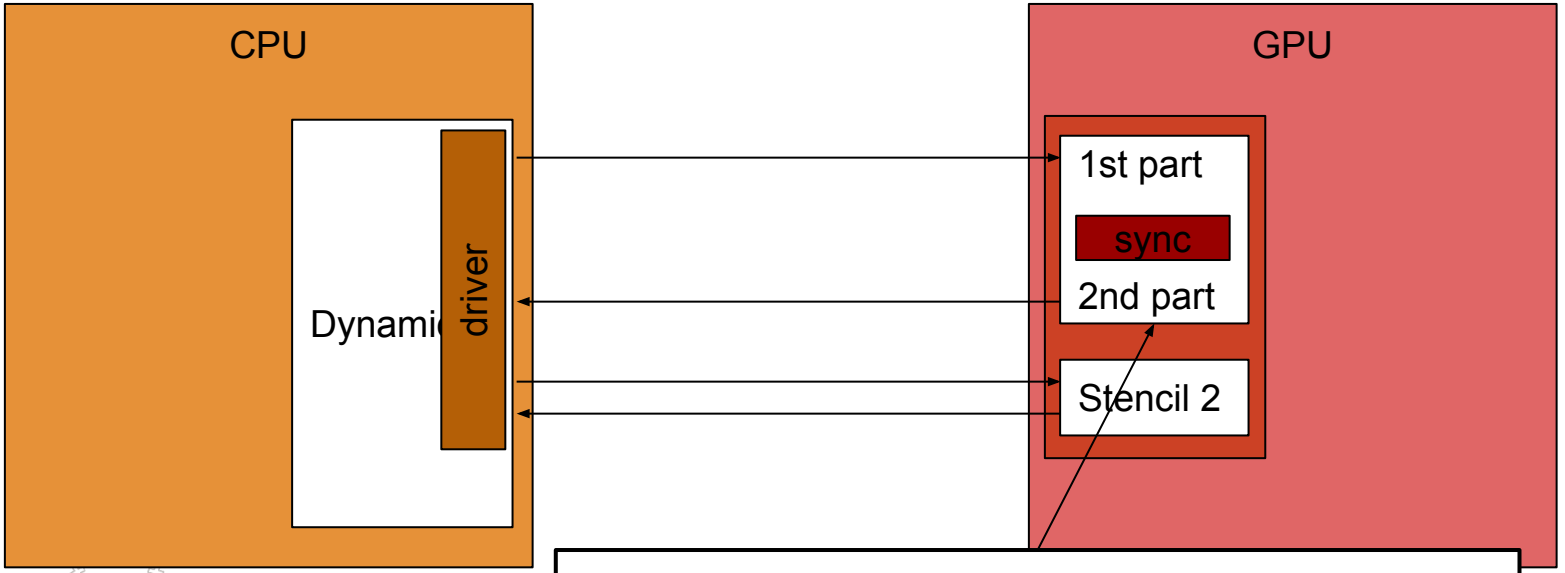
Dusk in weather & climate models



Dusk in weather & climate models

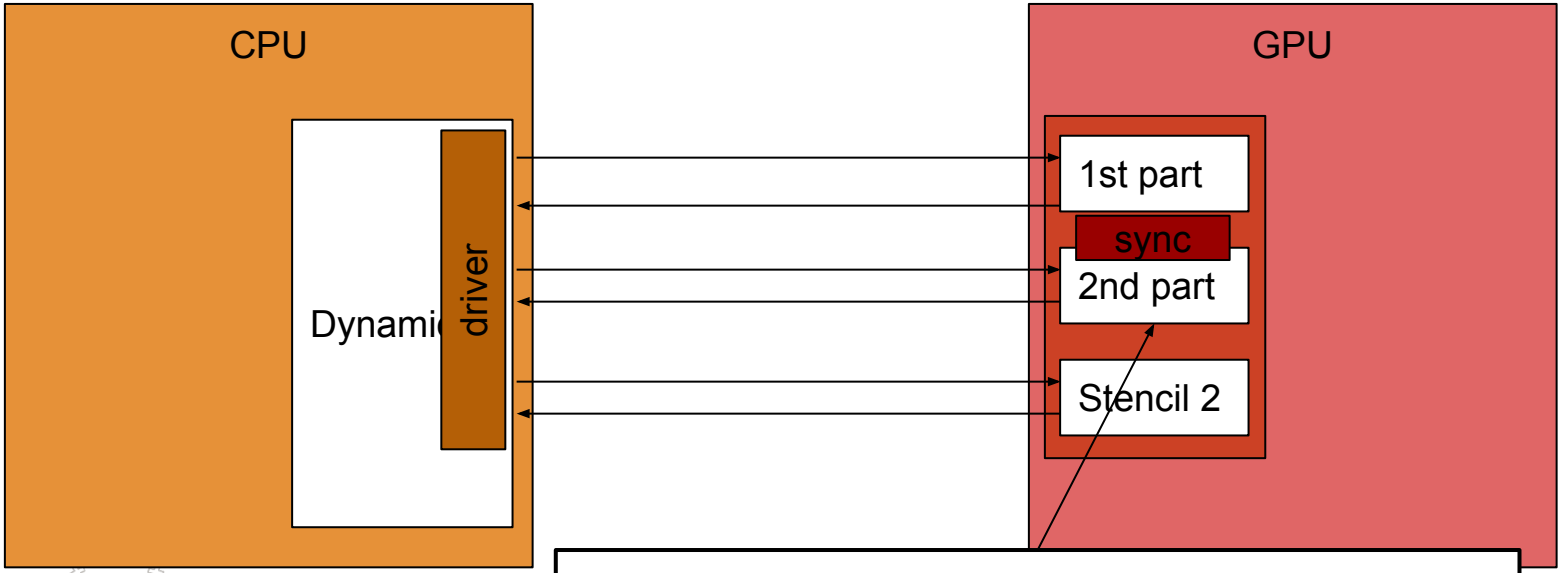


Dusk in weather & climate models



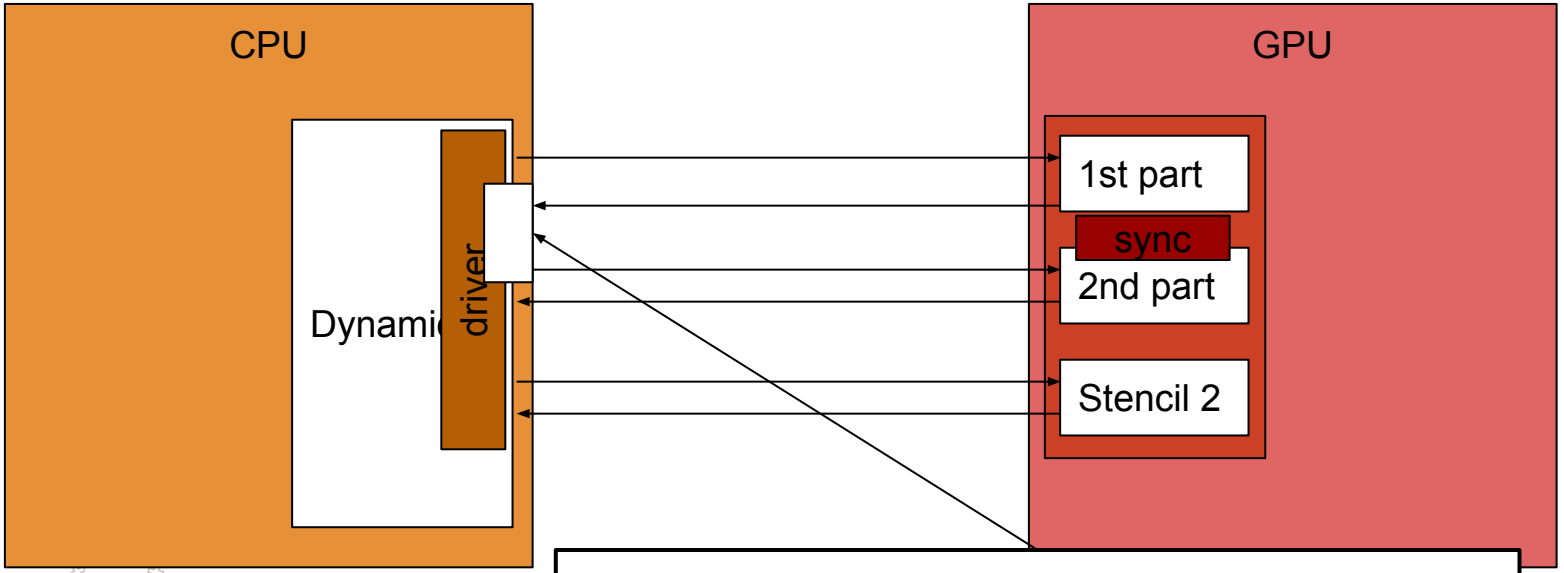
E.g., a bigger stencil consisting of two parts that requires a global synchronization in the middle.

Dusk in weather & climate models



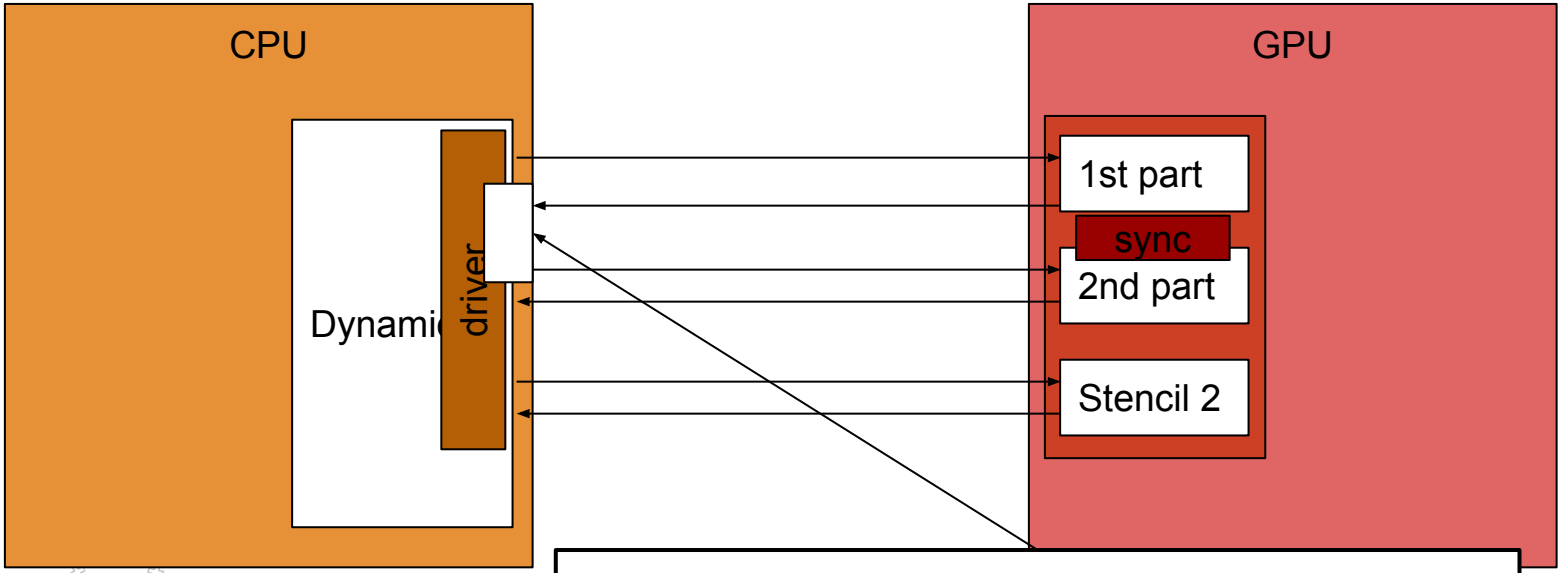
E.g., a bigger stencil consisting of two parts that requires a global synchronization in the middle.
Can only be done in CUDA by splitting the kernel.

Dusk in weather & climate models



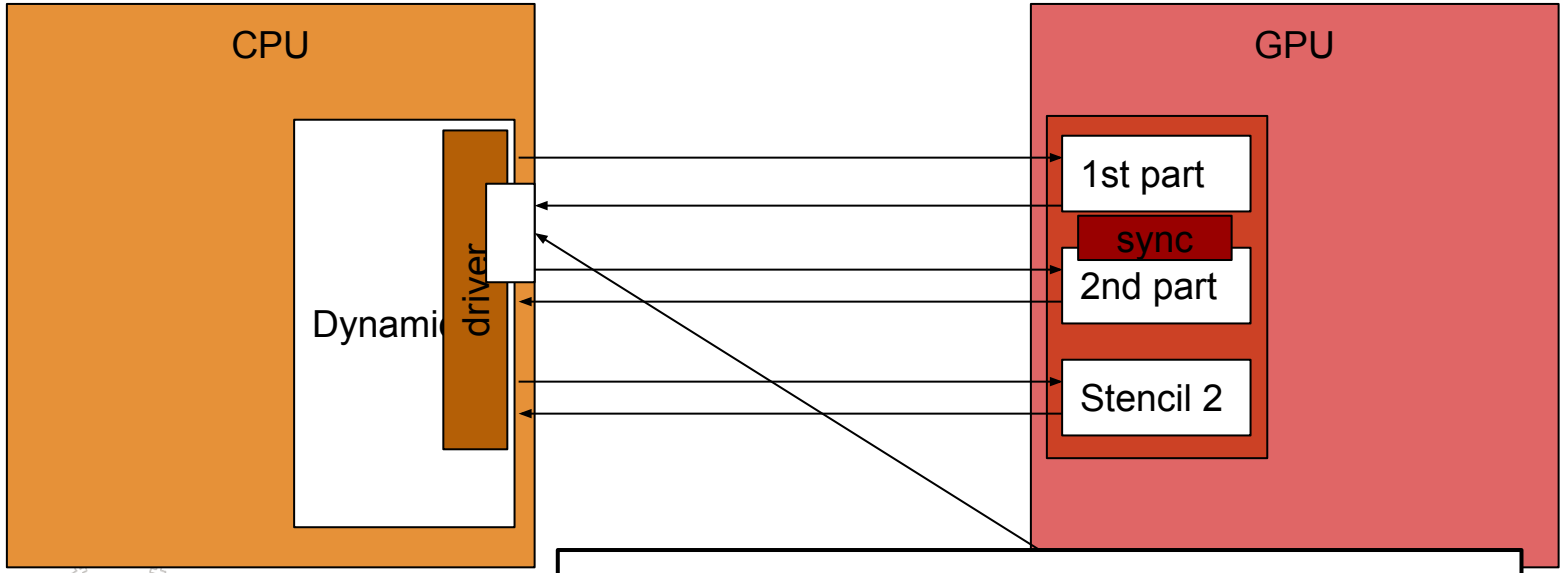
There's now a small part running on the CPU to make sure the 2nd part is called immediately after the 1st part. This is technically still part of the stencil.

Dusk in weather & climate models



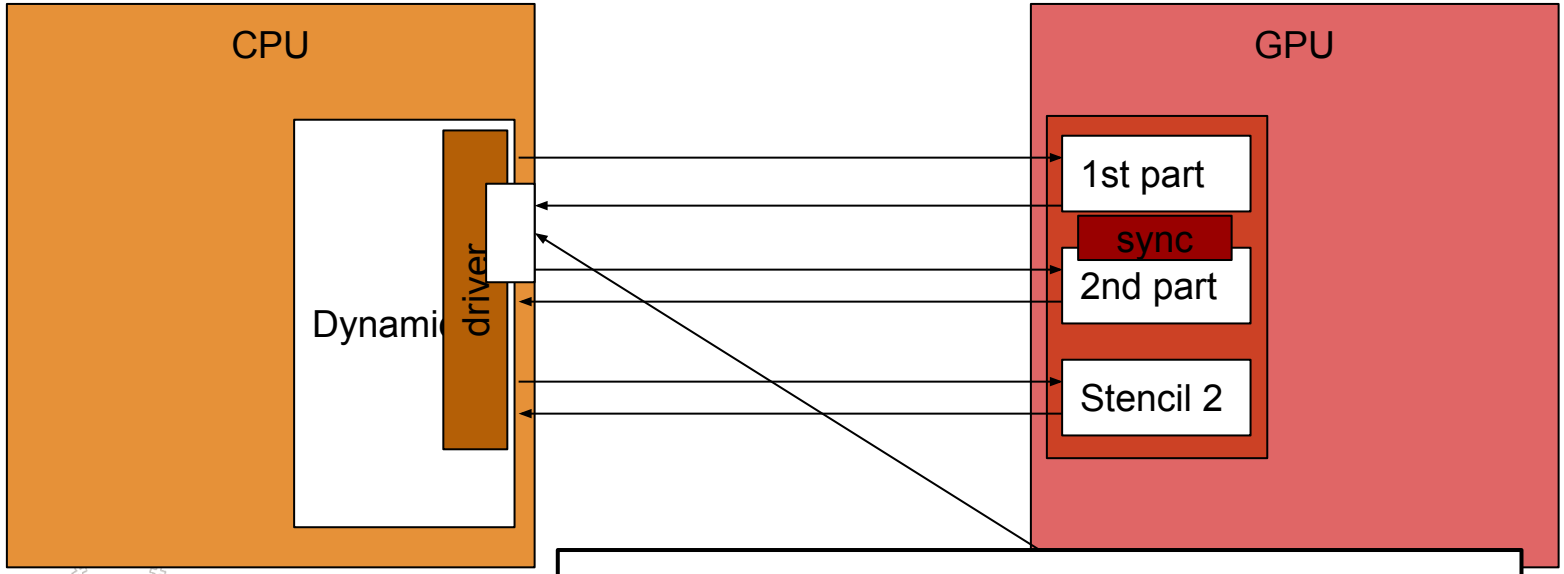
There's now a small part running on the CPU to make sure the 2nd part is called immediately after the 1st part. This is technically still part of the stencil.
It *drives* the CUDA kernel.

Dusk in weather & climate models



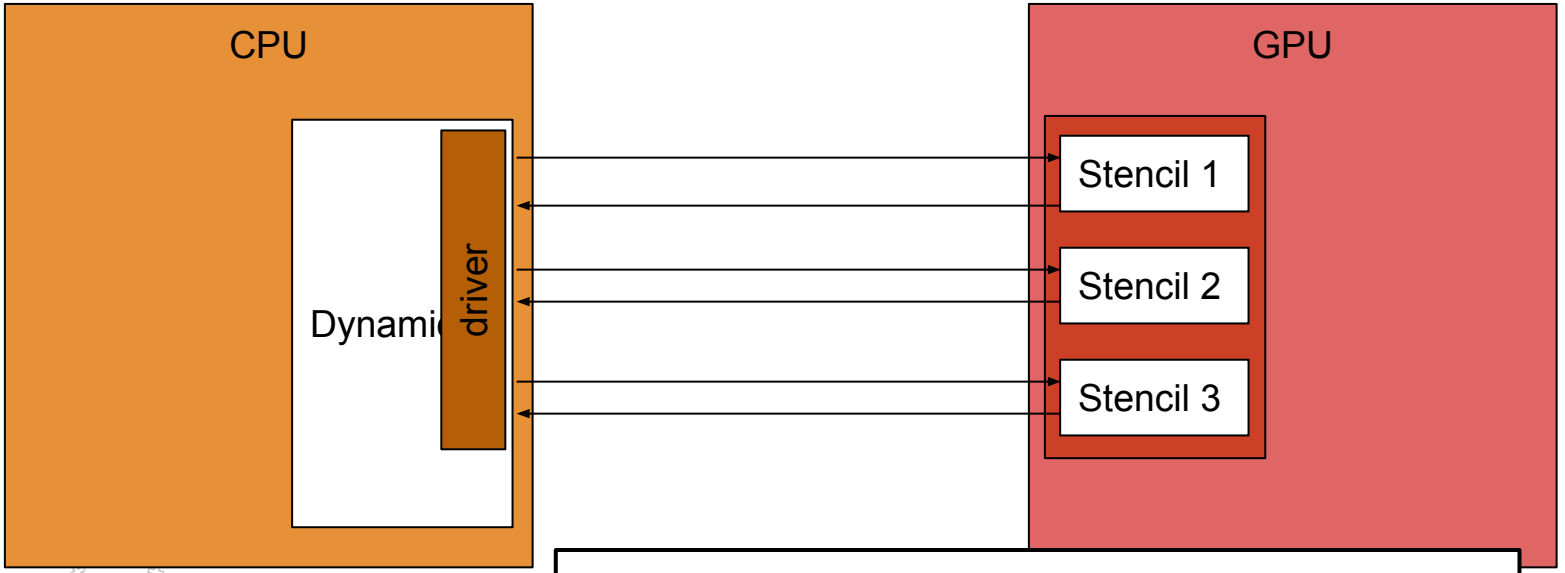
It *drives* the CUDA kernel.

Dusk in weather & climate models



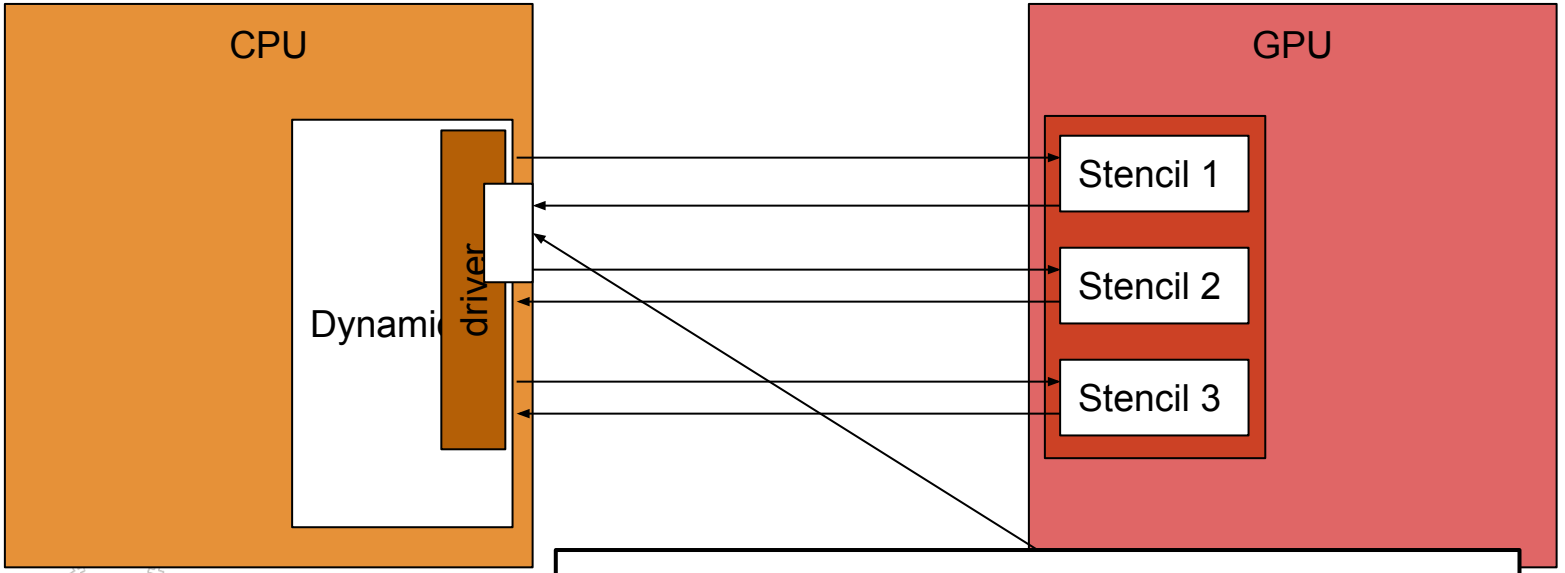
It *drives* the CUDA kernel.
But we don't consider it *driver code*.

Dusk in weather & climate models



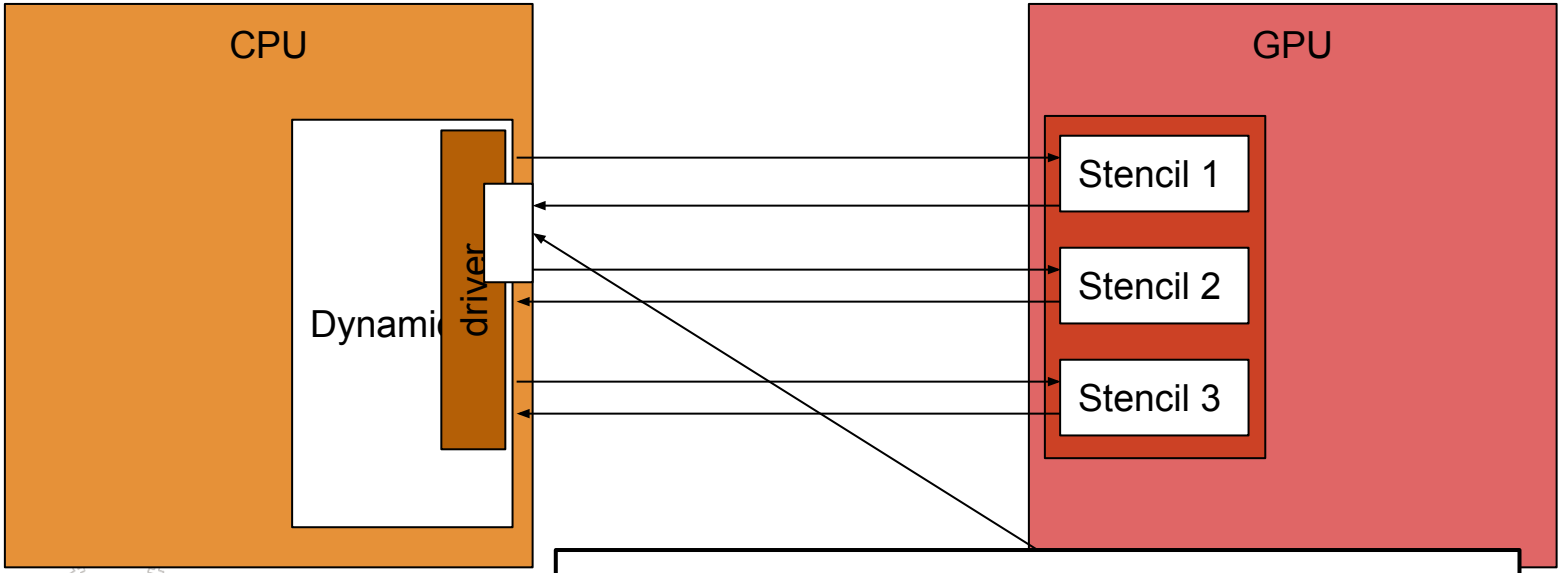
Another examples:

Dusk in weather & climate models



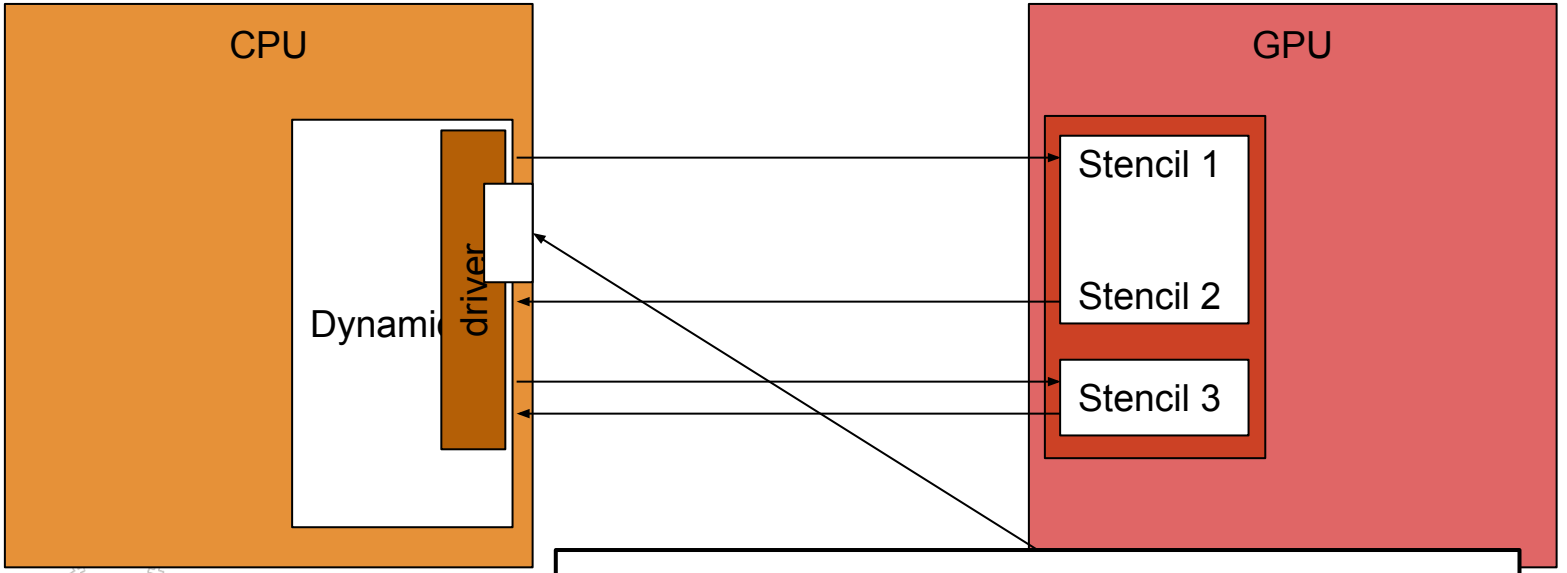
Another examples:
Simple driver code between two stencils.

Dusk in weather & climate models



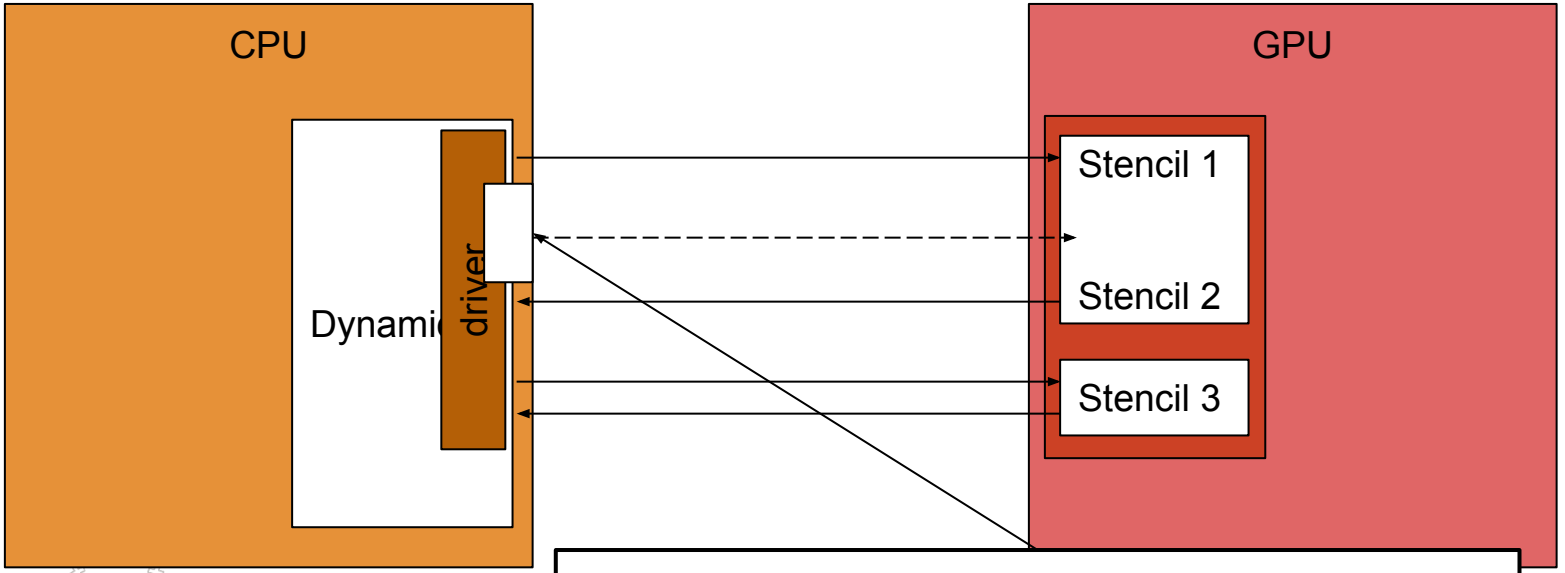
Another examples:
Simple driver code between two stencils.
We want to combine Stencil 1 & Stencil 2 to get better performance.

Dusk in weather & climate models



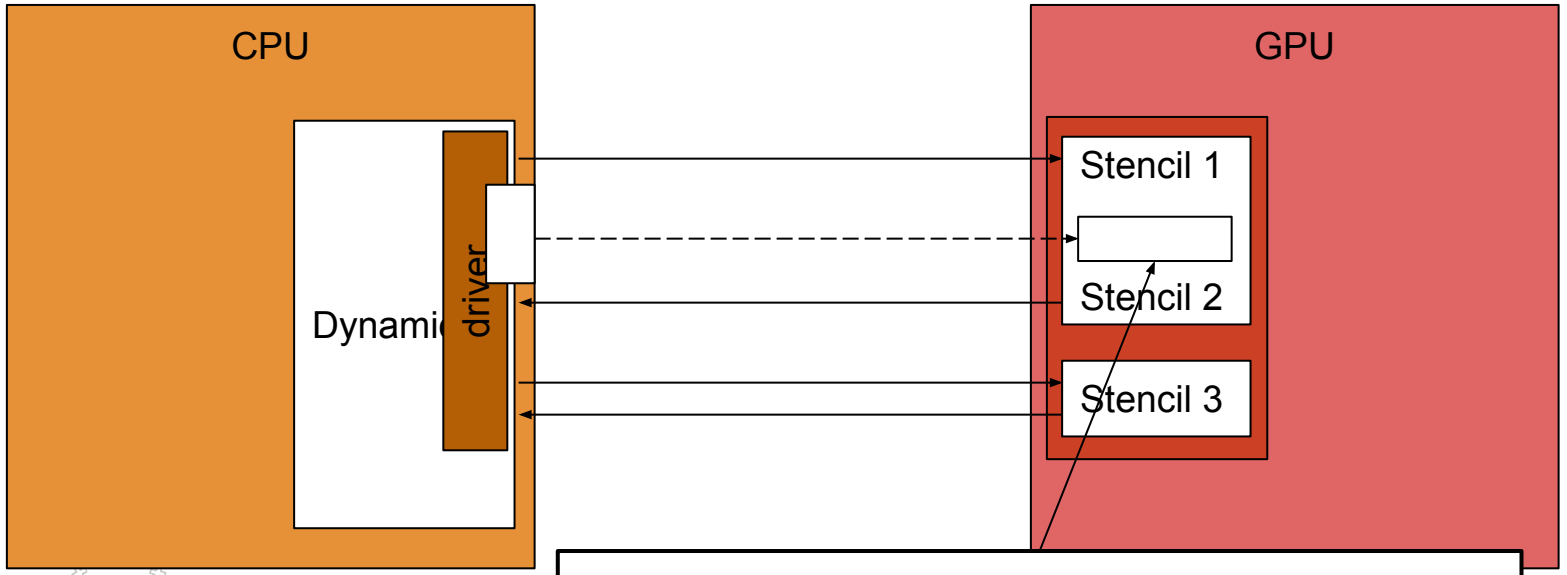
Another examples:
Simple driver code between two stencils.
We want to combine Stencil 1 & Stencil 2 to get better performance.

Dusk in weather & climate models



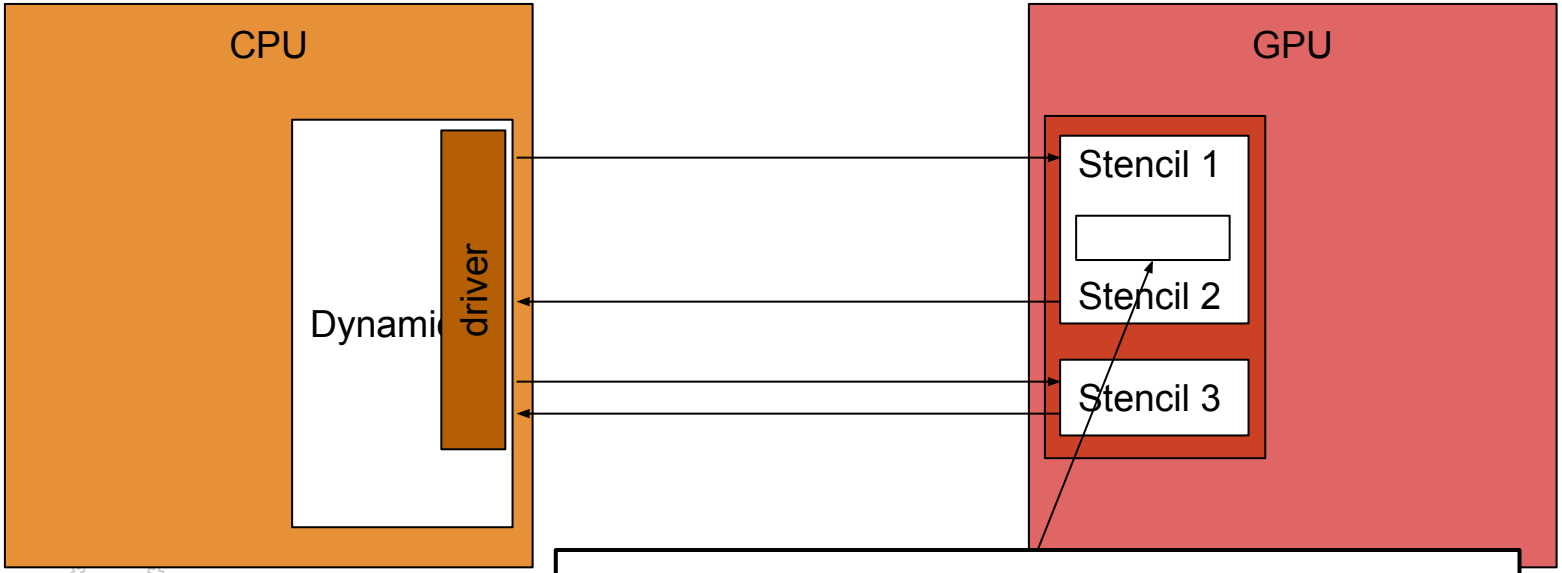
We let the previous *driver code* run on the GPU too.

Dusk in weather & climate models



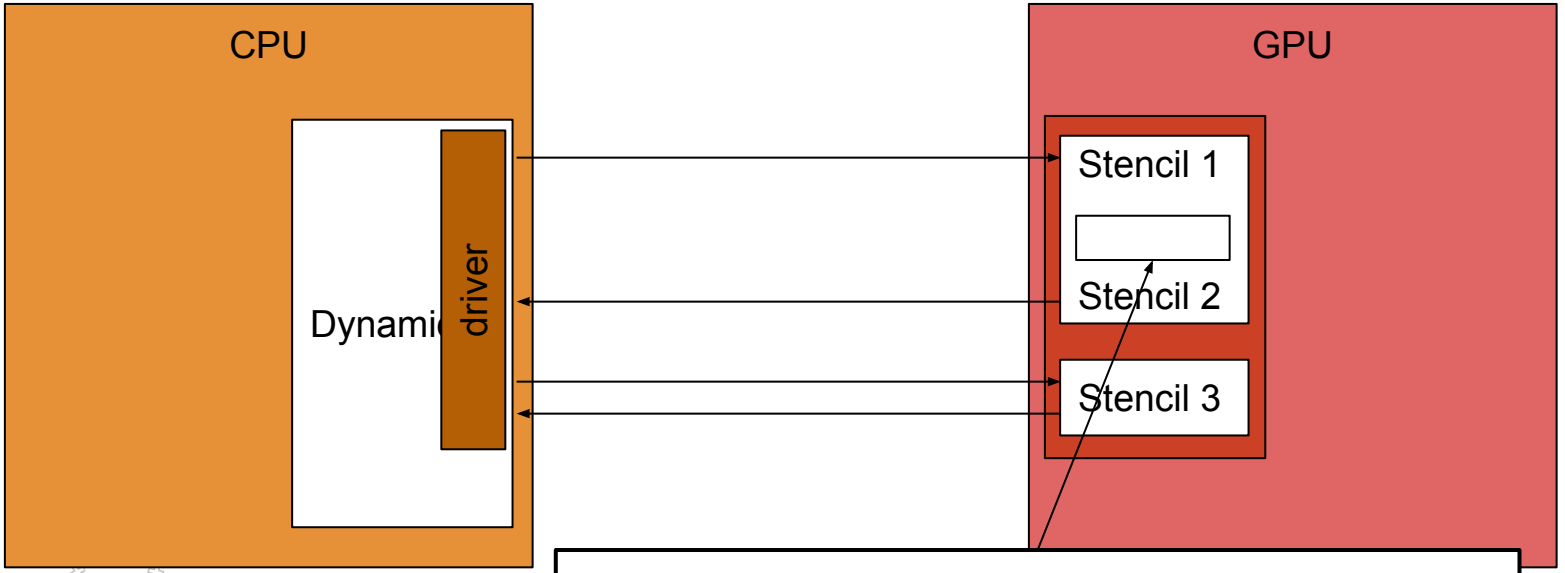
We let the previous *driver code* run on the GPU too.

Dusk in weather & climate models



We let the previous *driver code* run on the GPU too.

Dusk in weather & climate models



We let the previous *driver code* run on the GPU too. What was previously *driver code* has now become part of the CUDA kernel.

Dusk in weather & climate models

- Dusk is mostly the stencils
- Dusk requires driver code that
  - decides how/when the stencils are run
  - sets up the mesh
  - allocates memory
- The boundary between dusk & driver code is sometimes blurry
- Simple driver code that can run on the CPU can also be written in Dusk
  - In the future this will likely grow
  - Stencils can be merged into bigger stencils
    - More optimization opportunities

Dusk Stencil Language

- Dusk is a subset of Python (on the syntactic level)
- Dusk is statically typed
  - Python has dynamic types
  - Python added optional type hints in version 3.5 (2015)
  - These type hints are mandatory & enforced in dusk
- Dusk assigns different meanings to Python language elements
  - (different on the semantic level)

Dusk Stencil Language

- Python can do many things. E.g., starting a web server
  - We don’t want to start web servers in stencils
  - A lot of Python language elements are disallowed in dusk
- We assign different meanings to some Python elements
  - This is done to make dusk more *ergonomic*
  - Dusk targets domain scientists possibly with a Fortran background
  - It may upset experienced Python developers
  - This was a deliberate compromise

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Dusk eDSL

```
from dusk.script import *
```

Import *magic keywords* for the dusk language

```
@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Marks that a Python function is a dusk stencil

- This is what dusk will translate
- Other Python code will be ignored

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Name of the stencil

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Field definitions

More on this later...

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Apply the stencil on the whole domain
More on this later...

Dusk eDSL

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

What the stencil should compute

Extents & Neighborhoods will come later...

Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Sometimes we want to store an intermediate result temporarily.

Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    with levels_upward:
        a = b + 1
```

Sometimes we want to store an intermediate result temporarily.

For this we have temporary fields.

Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
```

Sometimes we want to store an intermediate result temporarily.

For this we have temporary fields.

We have to declare them at the lowest indentation level of the stencil.

Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Sometimes we want to store an intermediate result temporarily.

For this we have temporary fields.

We have to declare them at the lowest indentation level of the stencil.

Then we can use them for our stencil computations.

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

We distinguish between:

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

We distinguish between:

- *API Fields*

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

We distinguish between:

- *API Fields*
- And temporary Fields

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

API Fields:

- Are allocated/deallocated by the driver code

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

API Fields:

- Are allocated/deallocated by the driver code
- Persist between stencil runs

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

API Fields:

- Are allocated/deallocated by the driver code
- Persist between stencil runs
- Are *owned* by the driver code

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Temporary Fields:

- Are allocated/deallocated by the generated stencil

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Temporary Fields:

- Are allocated/deallocated by the generated stencil
- Only *exist* while the stencil is running

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Temporary Fields:

- Are allocated/deallocated by the generated stencil
- Only *exist* while the stencil is running
- Are *owned* by the stencil

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Temporary Fields:

- Are allocated/deallocated by the generated stencil
- Only *exist* while the stencil is running
- Are *owned* by the stencil

Dawn is free to remove or add temporary fields during optimization passes.

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Temporary Fields:

- Are allocated/deallocated by the generated stencil
- Only *exist* while the stencil is running
- Are *owned* by the stencil

Dawn is free to remove or add temporary fields during optimization passes.

However, dawn must respect and preserve how API Fields are mutated.

API fields vs Temporaries

```
from dusk.script import *

@stencil
def example_stencil(
    a: Field[Edge, K],
    b: Field[Edge, K]
):
    temp: Field[Edge, K]
    with levels_upward:
        a = b + 1
        temp = a + b
```

Unfortunately, user defined temporary fields are currently broken (only in the naive/CPU backend).

Don’t use them when solving the exercises.

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

Contextual information is automatically filled with sane defaults
-> Less verbose, more *to the point*

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

#  ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

This contextual information is mostly redundant and can sometimes become pretty big (redundant ≈ can be automatically inferred)

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

Code clones are generally avoided

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

But those aren't exactly code clones...

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

But those aren't exactly code clones…
True. However, we can infer that as well…
Because we describe the algorithm on a higher level

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

More on `sum_over` later

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

We also use high-level constructs to describe local neighborhoods

Dusk/Fortran Comparison

```
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

More on `Edge > Cell` later

Dusk/Fortran Comparison

```python
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```fortran
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

Still, we can appreciate similarities between Python and Fortran 👍

Dusk/Fortran Comparison

```python
if mask[k] or mask[k + 1]:
  a = sum_over(Edge > Cell, b - c)

  if a ⩽ d / e:
    f = g + max(
      3.141 * d + h,
      sqrt(a) / h + e ** h)

# ...
```

```fortran
IF (mask(k) .OR. mask(k + 1)) THEN
  a = b(j,1,blk) - &
    c(nbh(j,blk,1),k,nbh_blk(j,blk,1)) + &
    b(j,2,blk) - &
    c(nbh(j,blk,2),k,nbh_blk(j,blk,2))

  IF (a ⩽ d / e(j,k,blk)) THEN
    f = g + MAX(3.141_wp * d + h, &
      SQRT(a) / h + e(j,k,blk) ** h)

! ...
```

Still, we can appreciate similarities between Python and Fortran 👍
(for the exercises, if you want to do `x ** 2` you have to write that as `x ** 2.0` due to a small typing issue)

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

This is a bigger neighborhood that contains 4 neighboring *locations*

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

During compilation this can be automatically unrolled

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

We concisely describe:

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

We concisely describe:

- What we compute

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

We concisely describe:

- What we compute
- Where we compute it

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

Both are separated ≈ not interleaved

Dusk/Fortran Comparison

```
m = sum_over(Edge > Cell > Vertex, o * p)
```

```
m(j,k,blk) = &
  o(j,1,blk) * &
    p(nbh_p(j,blk,1),k,blk_p(j,blk,1)) + &
  o(j,2,blk) * &
    p(nbh_p(j,blk,2),k,blk_p(j,blk,2)) + &
  o(j,3,blk) * &
    p(nbh_p(j,blk,3),k,blk_p(j,blk,3)) + &
  o(j,4,blk) * &
    p(nbh_p(j,blk,4),k,blk_p(j,blk,4))
```

Yes, not all code samples will see such a big change.
But these benefits are very real and lead to (arguably) better code overall.

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

A more *moderate* example

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

Dusk doesn't magically remove all redundancies

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

Dusk doesn't magically remove all redundancies
It's important to acknowledge this for an honest and fair discourse

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

This introduces an element-wise multiplication with a *weights vector*.
For some stencils this is necessary.
(More on this later)

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

Overall, the dusk version is expressed in a more *high-level* way

Dusk/Fortran Comparison

```
a = 4.0 * sum_over(
    Edge > Cell > Vertex,
    b + c,
    weights=[
        d ** 2,
        d ** 2,
        e ** 2,
        e ** 2,
    ],
)
```

```
a(je,jk,jb) = 4._wp * ( &
  (b4(je,jk) + b3(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *e(je,jb)**2 + &
  (b2(je,jk) + b1(je,jk) + &
    2._wp*c(je,jk,jb)) &
    *d(je,jb)**2 )
```

Dusk naturally has an edge over Fortran, because they target different domains:

- Dusk: Specific to climate and weather models
- Fortran: More general, for numeric computation and scientific computing

Dusk General Info

- A minimal and lightweight front-end for dawn
  - (Has been growing steadily)
- Started May 2020
- 2.6k LoC Pure Python
- No dependencies
  - Only Python's standard library
    - (We might add small dependencies in the future)
- Currently only works with Python 3.8
  - (Can be changed)
- Most functionality is a rather light translation pass from Python AST to SIR
  - Very close to SIR for unstructured meshes

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

MeteoSwiss
Operation Center 1
CH-8058 Zurich-Airport
T +41 58 460 91 11
www.meteoswiss.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l'Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch