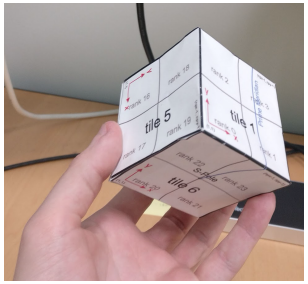
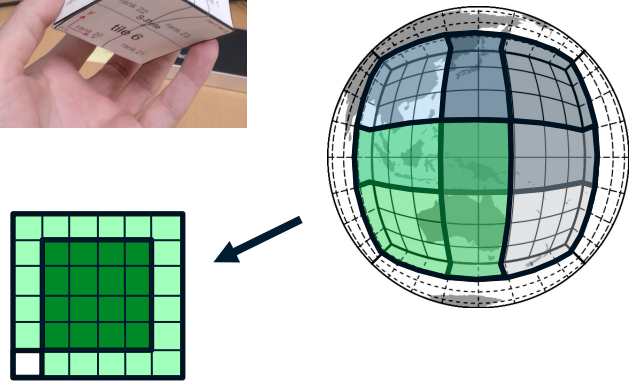
Workshop on

Domain-Specific Languages for Performance-Portable Weather and Climate Models





Content: Infrastructure and Halo Updates

Presenter: Jeremy McGibbon

Motivation

So you've learned everything about GT4py.

You want to build a dynamical core.

What's missing?

(type in Slack)

Motivation

So you've learned everything about GT4py.

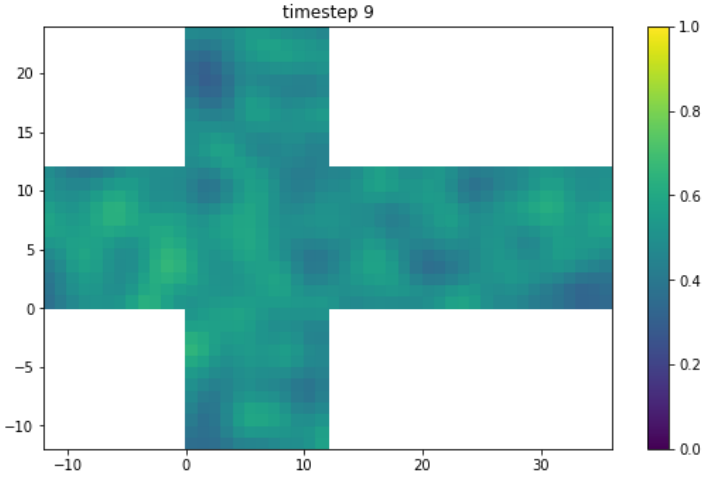
You want to build a dynamical core.

What's missing?

- Domain decomposition
- Metadata
- Halo updates

What to expect in this session

- Start with motivation
- Material:
  - Partitioning the cube
  - Quantity
  - Halo updates
- End goal: run a gt4py stencil in parallel on the cubed sphere



Motivation

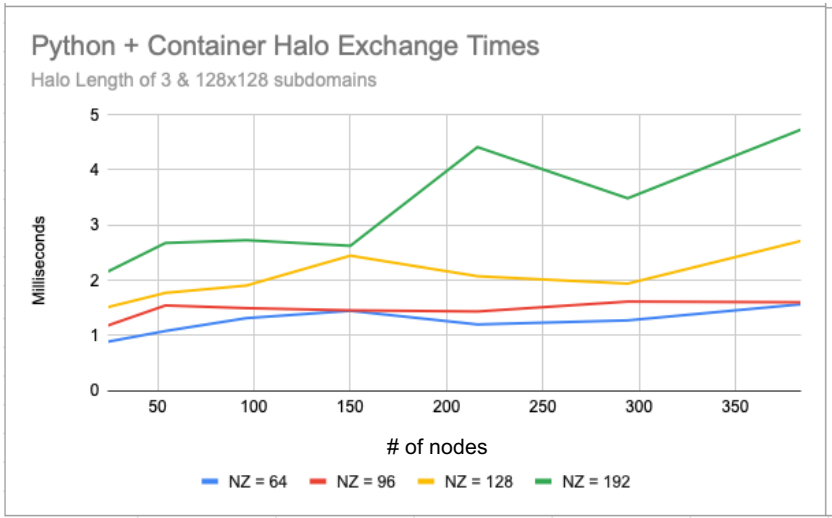
FMS: To wrap or not to wrap

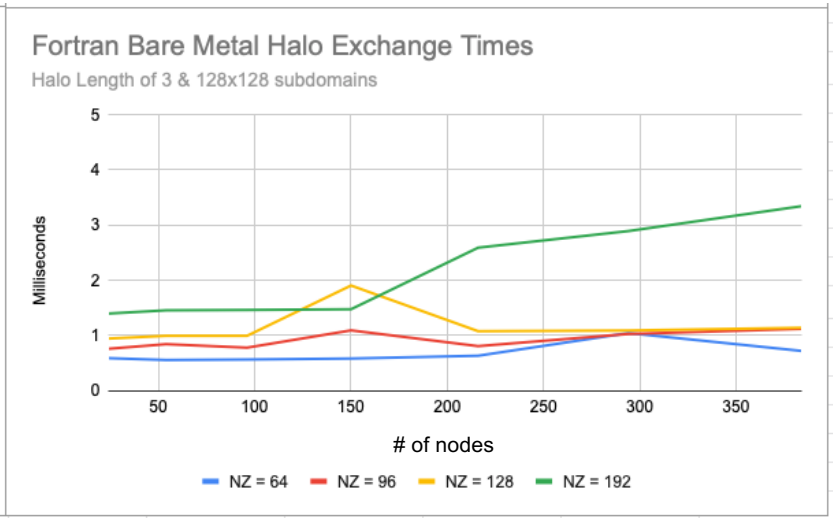
- FMS already provides performant halo updates, could be wrapped
- Would be an unknown challenge to get FMS to work with GPU data
- Easier to modify code in a high-level dynamic language

Motivation

FMS Performance Comparison

- FMS more performant, but both scale well





Motivation

What is fv3gfs-util?

- Used by dynamical core and by fv3gfs-wrapper (workshop in Jan)
- Must support both “model” code like dynamics and high-level analysis code
- FMS is a framework, fv3gfs-util is a toolkit

```
cube.halo_update(quantity, n_points=3)
```

Learning goals for this session

- Understand how data is partitioned between processes on the FV3 cubed sphere
    - Know what a 3D numpy array represents as part of a cubed sphere
- Track halo and non-halo data using Quantity
- Perform a halo update on the cubed sphere using MPI

The FV3 Cubed Sphere

Learning Goal*:

- How is data partitioned between processes on the FV3 cubed sphere?
    - How much of the globe is on one process?

* this review will motivate the existence of code objects

The FV3 Cubed Sphere

Dynamics

- Shallow water (2D) advection in the horizontal
- Lagrangian advection in the vertical
    - i.e. thickness of shallow layers changes
- Periodic remapping in the vertical

The FV3 Cubed Sphere

Horizontal Grid

- Maps a cube to a sphere
  - Much easier to partition a cube
- Take a cube, and “inflate” it outward to the planetary surface

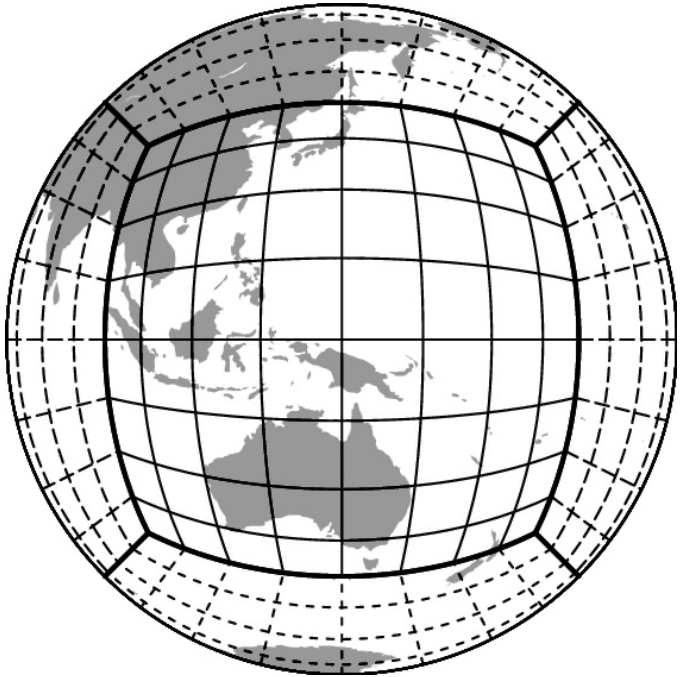
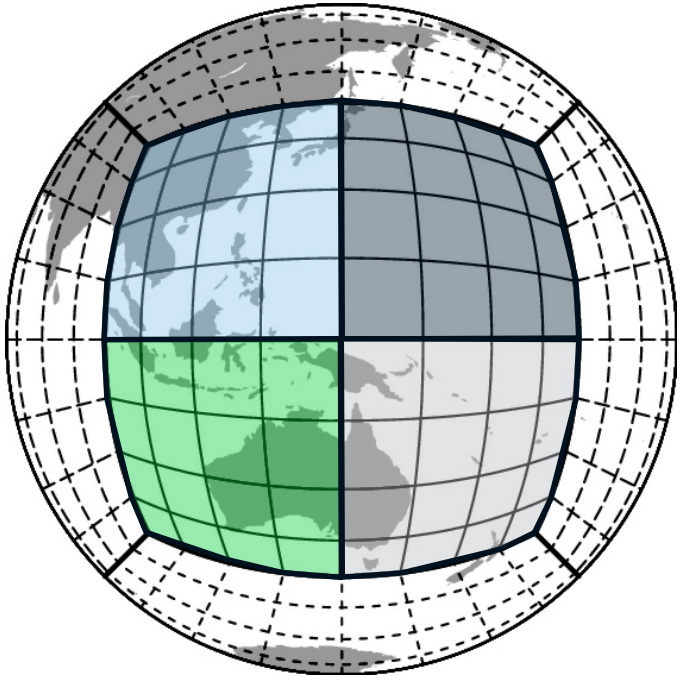


Figure 5 from Washington et al. 2008

The FV3 Cubed Sphere

Partitioning

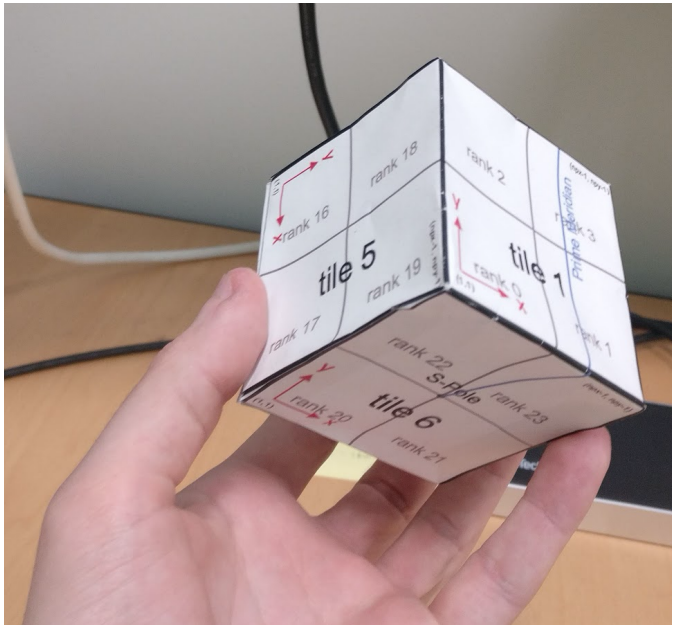
- Each tile face is separated into processes (ranks) according to the same “layout” (nx, ny)
- e.g. a (2, 2) layout has 4 processes on each tile, and 24 processes in total



The FV3 Cubed Sphere

Partitioning

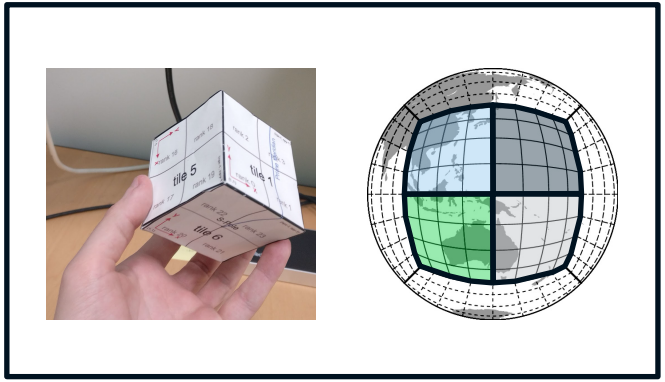
- Given a rank and a direction, tell me:
  - What rank is over there?
  - What is the relative rotation of the two ranks?
- Complex responsibility!
  - Needs its own encapsulation



The FV3 Cubed Sphere

Partitioning

```
>>> import fv3gfs.util

>>> layout = (2, 2)

>>> partitioner = fv3gfs.util.CubedSpherePartitioner(

...     fv3gfs.util.TilePartitioner(layout)

... )

>>> print(partitioner.boundary(fv3gfs.util.WEST, rank=0))

SimpleBoundary(from_rank=0, to_rank=19, n_clockwise_rotations=1,
boundary_type=0)
```



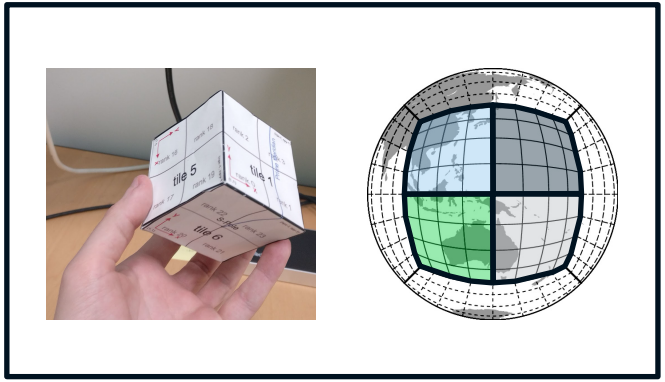
CubedSpherePartitioner

- Separate concerns of how data is distributed within a tile from how tiles are numbered on the cube

The FV3 Cubed Sphere

Partitioning

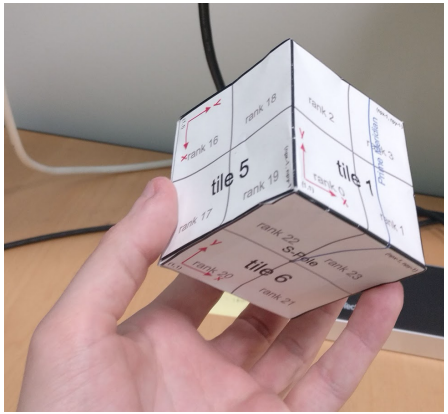
```
>>> import fv3gfs.util

>>> layout = (2, 2)

>>> partitioner = fv3gfs.util.CubedSpherePartitioner(

...     fv3gfs.util.TilePartitioner(layout)

... )

>>> print(partitioner.boundary(fv3gfs.util.WEST, rank=0))

SimpleBoundary(from_rank=0, to_rank=19, n_clockwise_rotations=1,
boundary_type=0)
```



CubedSpherePartitioner

- Encapsulates logic for spatial/orientation relationships between ranks
- Not associated with one particular rank

Notebook: Partitioner



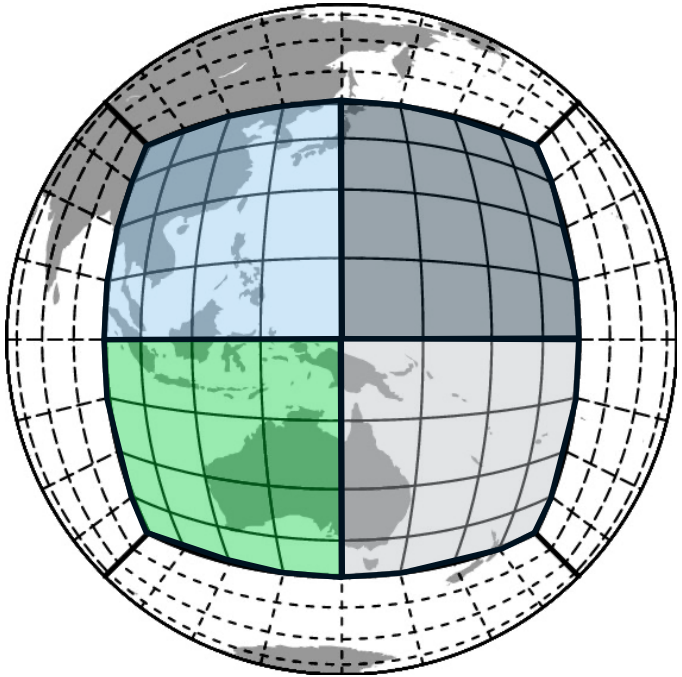
See you on Slack! **Return in 10 minutes at 1:55 EST.**

The FV3 Cubed Sphere

Partitioning

We asked: How is data partitioned between processes on the FV3 cubed sphere?

Do the shaded regions represent the numpy array on each process?

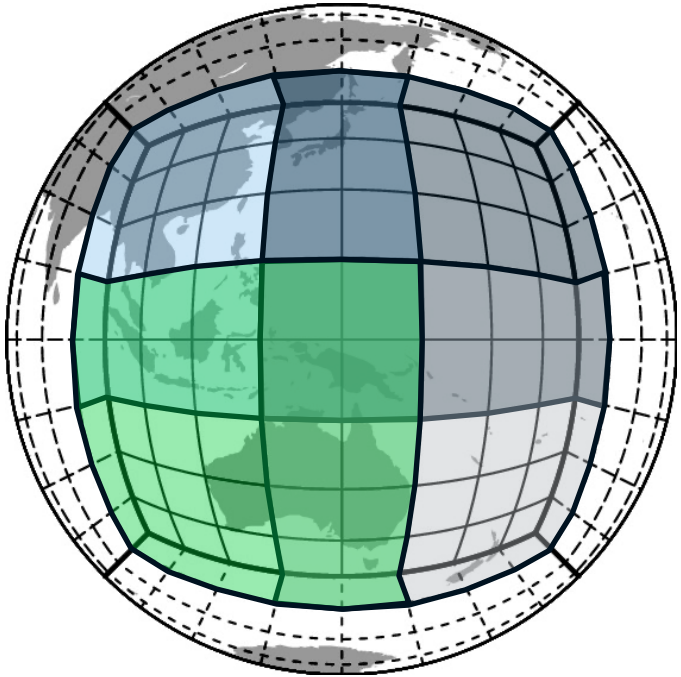


The FV3 Cubed Sphere

Partitioning

We asked: How is data partitioned between processes on the FV3 cubed sphere?

Next up: Halos on the cubed sphere



Halos in the FV3 Cubed Sphere

Learning Goal*:

- What do halos look like on the cubed sphere, especially on tile corners?

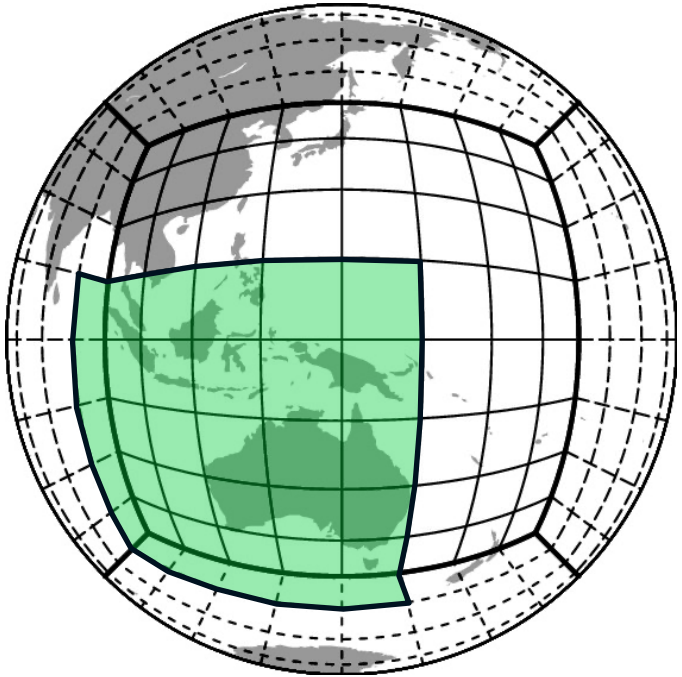
* this review will motivate the existence of code objects

Halos in the FV3 Cubed Sphere

Halos

- Must extend inputs to include data from neighboring domains required for spatial derivatives

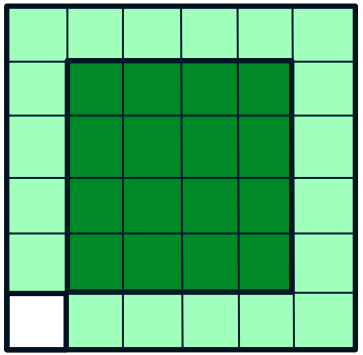
- Outputs are not extended to include the halo (if you did, you'd need to even further extend the inputs)

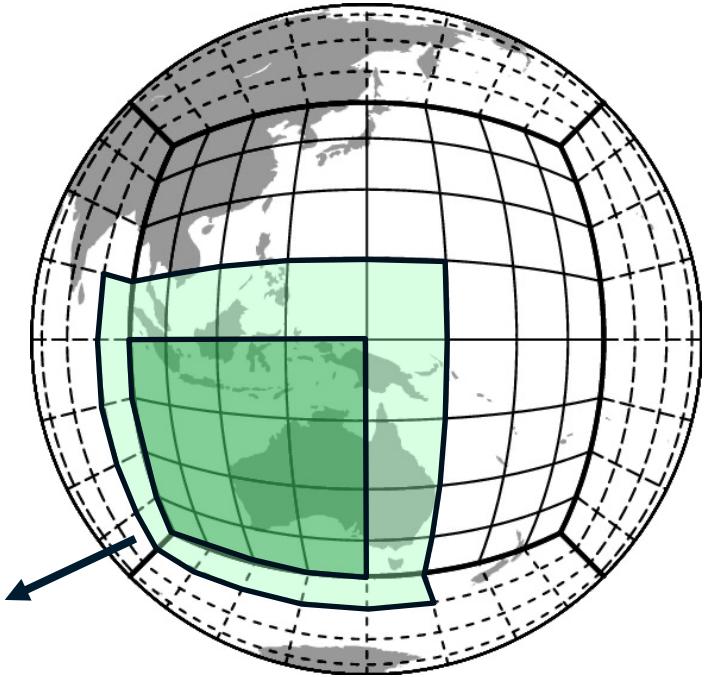


Halos in the FV3 Cubed Sphere

Halos

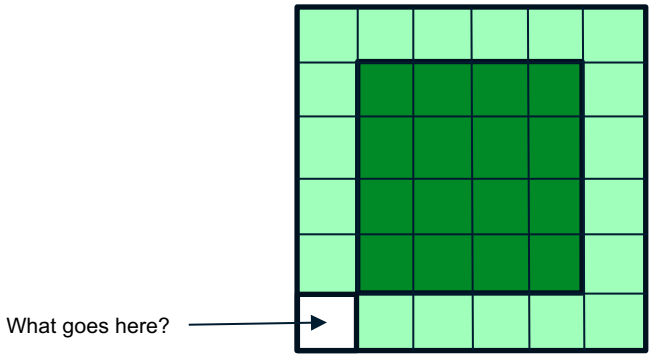
- Input data from neighboring ranks required to compute output data on this rank
- Not part of the “compute domain” means this rank is not responsible for it

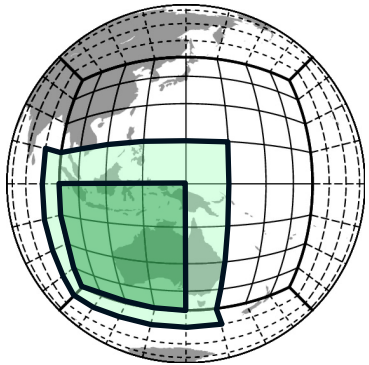




Halo Corners : Discussion

(type in Slack)

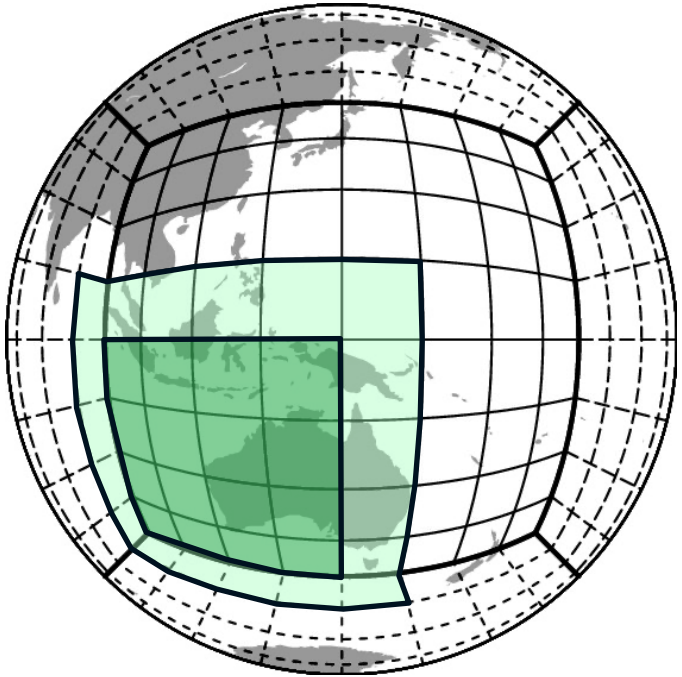




Halos in the FV3 Cubed Sphere

Halos on Subtile Interfaces

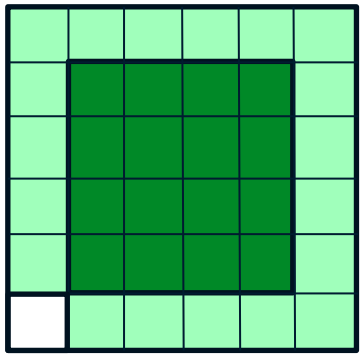
- Interface variables are a little strange
- Included in compute domain of both neighbor subtiles
- Must be periodically synchronized



Halos in the FV3 Cubed Sphere

Indexing Halos

- Earlier have seen (nhalo, nhalo, 0)
- Numpy* does not support offset indexing, so (0, 0) is the corner of the array on this rank
- Means your compute domain indices are n_halo + I
    - And n_halo can be different for different variables!

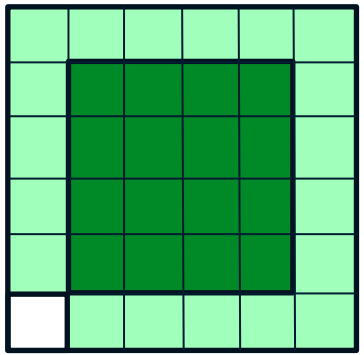


* In Fortran FV3, array indexes are offset to use global indexing so (0, 0) is the corner of the compute domain of the tile face

Halos in the FV3 Cubed Sphere

Quantity

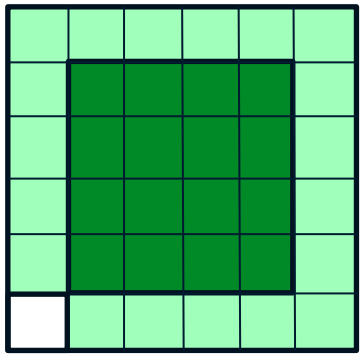
- Numpy array wrapped with metadata
  - origin
  - extent (domain) → Define the halo
  - dims
  - units
- `quantity.data` is the numpy array
- `quantity.view[:]` gives a view of just the compute domain



Halos in the FV3 Cubed Sphere

Why not Storage?

- Different concerns
- GT4Py has no halo updates
- Quantity is used outside of dycore
  - e.g. by fv3gfs-wrapper
- Carries metadata needed for IO
- Self-documenting code



Quantity

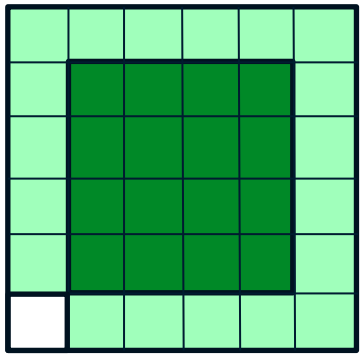
Learning Goals:

- Initialize a Quantity from a numpy array or gt4py storage
- Retrieve a gt4py storage from a Quantity

Quantity

Quantity Initialization

```python
import numpy as np
import fv3gfs.util

array = np.zeros([6, 6])
quantity = fv3gfs.util.Quantity(
    array,
    origin=(1, 1),
    extent=(4, 4),
    dims=[fv3gfs.util.X_DIM, fv3gfs.util.Y_DIM],
    units="degK"
)
```

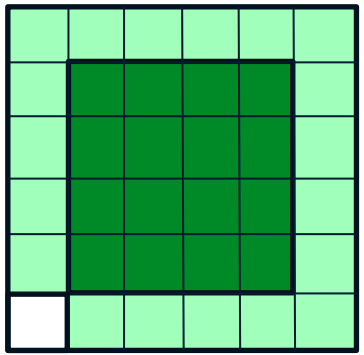


- origin and extent are optional, but needed if you want halos

Quantity

GT4Py Integration

```python
import fv3gfs.util

quantity = fv3gfs.util.Quantity(
    array,
    origin=(1, 1),
    extent=(4, 4),
    dims=[fv3gfs.util.X_DIM, fv3gfs.util.Y_DIM],
    units="degK",
    gt4py_backend="numpy"
)

print(quantity.storage)  # allocates a gt4py storage
```

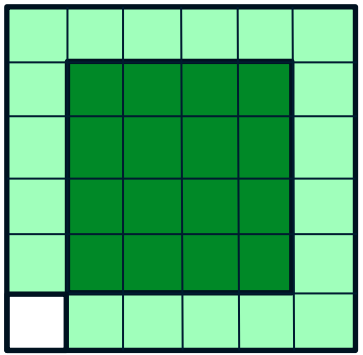


Can use this as a way to create gt4py storages

Quantity

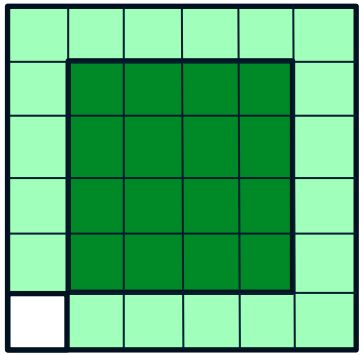
GT4Py Integration

```
import fv3gfs.util
import gt4py as gt

nhalo = 1
nx, ny = 4, 4
shape = (nx + 2*nhalo, ny + 2*nhalo)
origin = (nhalo, nhalo)
backend = "numpy"
field = gt.storage.zeros(backend, origin, shape, dtype=np.float64)

array = np.zeros([6, 6])
quantity = fv3gfs.util.Quantity(
    field,
    origin=origin,
    extent=(nx, ny),
    dims=[fv3gfs.util.X_DIM, fv3gfs.util.Y_DIM],
    units="degK"
)
```



Can initialize from gt4py storages

Notebook: Quantity



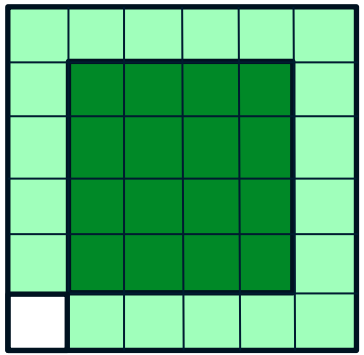
See you on Slack! **Return in 10 minutes at 2:20 EST.**

Quantity

Quantity GPU Integration

If a GPU backend or storage is used:

- `quantity.data` will be cupy array
- `quantity.np` will be cupy, normally exists as numpy
- Means e.g. `quantity.np.mean(quantity.data)` will work regardless of whether using cupy or numpy



Halo Updates

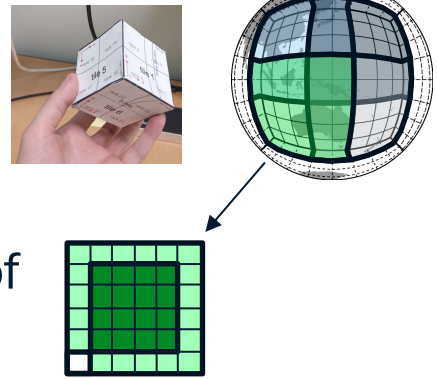
Learning Goal:

- Perform a halo update using a Quantity

Halo Updates

Review

- Partitioner gives info needed to match ranks for halo updates



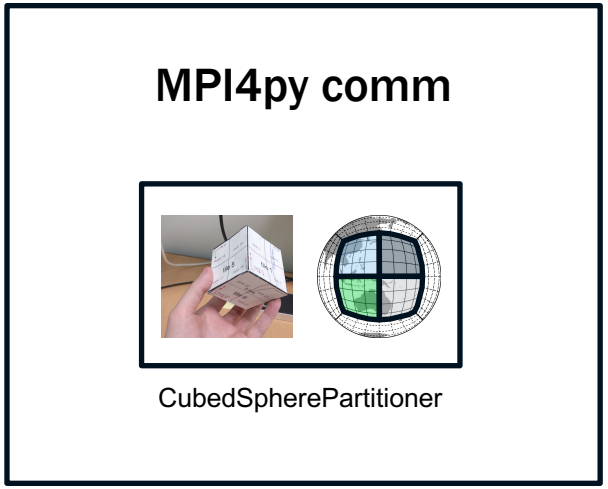
- Quantity marks index positions of halos

- We have all the information we need for a halo update!

Halo Updates

CubedSphereCommunicator

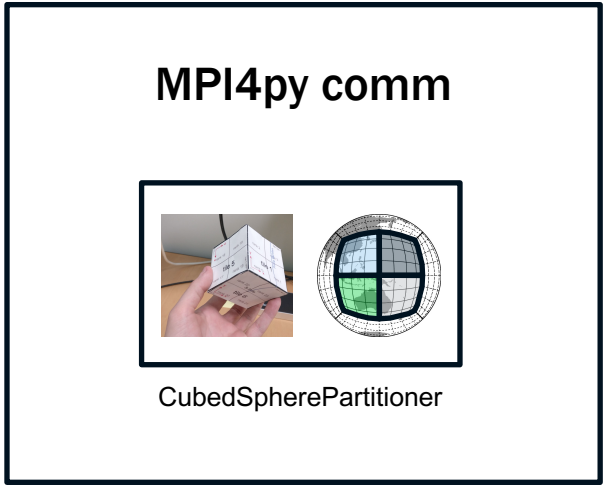
- Composed of a CubedSpherePartitioner and an MPI4py communicator



Halo Updates

CubedSphereCommunicator

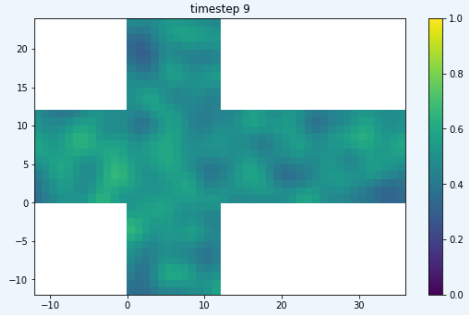
```
>>> import fv3gfs.util
>>> from mpi4py import MPI
>>> layout = (2, 2)
>>> partitioner = fv3gfs.util.CubedSpherePartitioner(
...     fv3gfs.util.TilePartitioner(layout)
... ) >>> cube = fv3gfs.util.CubedSphereCommunicator(
...     MPI.COMM_WORLD,
...     partitioner
... )
```



```
cube.halo_update(quantity, n_points=3)
```

Hands-on Session

Now it's your turn!



Session_3B_CubedSphereCommunicator.ipynb to work on

- Run your own stencil on the cube!
- Can return to "Moving Forward" sections in previous notebooks if you have time

See you on Slack! **Next huddle at 4:15 pm EST.**