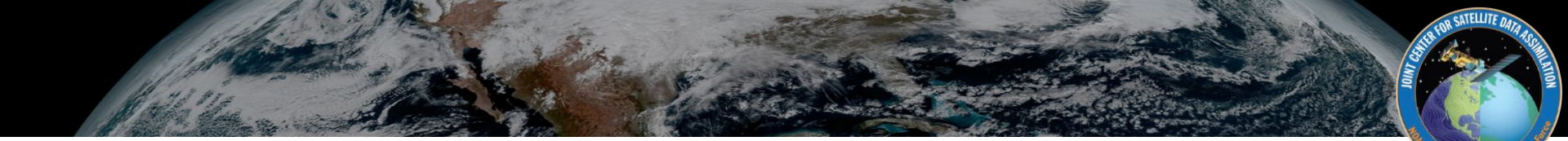


Updating atlas interface between JEDI and models

François Hébert — 10 Aug 2023 JEDI weekly meeting

Updating atlas interface

JEDI provides “generic” algorithms: interpolations, VADER, SABER, …

model data ↔ atlas data ↔ JEDI generic algorithms

Updating atlas interface

JEDI provides “generic” algorithms: interpolations, VADER, SABER, …

model data ↔ atlas data ↔ JEDI generic algorithms

Two wishlist items

- access more atlas functionality from within JEDI
  - atlas interpolations, halo exchanges, …
- document and clarify API
  - improve consistency across models
  - for JEDI team, how to develop and maintain generic code
  - for model teams, what to implement

Updating atlas interface

JEDI provides “generic” algorithms: interpolations, VADER, SABER, …

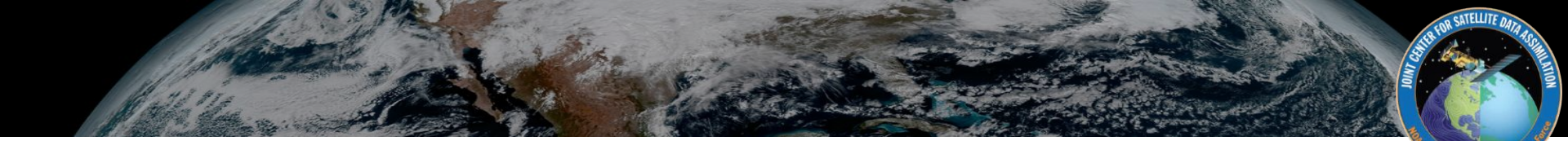
model data ↔ atlas data ↔ JEDI generic algorithms

In weeks ahead, we’ll update and formalize this API

- google doc with interface requirements
- same pattern as current atlas usage
  - more model geometry information passed into atlas

Today

- summary of interface updates
- logistics
- illustrations of trickier geometric parts



Summary of interface changes

Current atlas interface

```
// Geometry methods
const atlas::FunctionSpace& Geometry::functionSpace() const;
const atlas::FieldSet& Geometry::extraFields() const;

// State methods
void State::toFieldSet(atlas::FieldSet&) const;
void State::fromFieldSet(const atlas::FieldSet&);

// Increment methods
void Increment::toFieldSet(atlas::FieldSet&) const;
void Increment::fromFieldSet(const atlas::FieldSet&);
void Increment::toFieldSetAD(const atlas::FieldSet&);
```

Geometry atlas interface changes

```
const atlas::FunctionSpace& Geometry::functionSpace() const;
```

- add connectivity to Mesh in FunctionSpace
  - typically, by importing connectivity data from model into atlas
  - [see final section of slides]

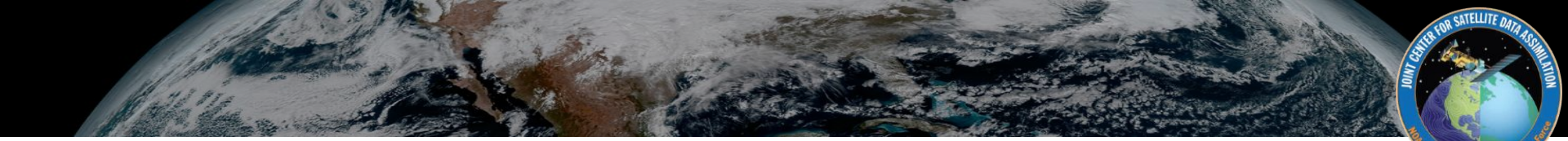
```
const atlas::FieldSet& Geometry::fields() const;
```

- rename
  - method: extraFields() to fields()
  - individual Fields: hmask to owned; vunit to vert_coord
- [TBD: may split this into multiple methods]

State/Increment atlas interfaces changes

```
void Increment::toFieldSet(atlas::FieldSet&) const;
void Increment::fromFieldSet(const atlas::FieldSet&);
```

- remove toFieldSetAD (handle adjoint accumulation in JEDI)
  - to/fromFieldSet become simple data copies
- reorder data in Fields to match FunctionSpace changes
  - building halos via atlas => easiest to fill halos via atlas's haloExchange
  - [if this is a problem, I can expand atlas to import model halos too]
- extra guidelines for various scenarios
  - halo contents
  - handling of missing fields
  - units [after CCPP variable naming and units adopted in JEDI]



Logistics

Suggested implementation sequence

Set up environment

- unload modules for atlas,eckit,fckit (fiat,ectrans?), compile develop in bundle
- C++17
- branch feature/atlas in OOPS and SABER [branches still WIP!]

Edit model interface classes to compile

- rename Geometry::extraFields()
- remove Increment::toFieldSetAD()

Refactor

- first, Geometry::functionSpace
- next, State/Increment::to/fromFieldSet
- use interface tests for Geometry/State/Increment as (partial) checks

Roadmap

[TBD] initial “easy” PR renaming Geometry fields method?

Main refactor of functionSpace and to/fromFieldSet

- aiming for Sept/Oct to enable subsequent developments
- depends on
  - spack-stack updates in mid-Sept
  - MPI work in atlas for coupled and EDA (timeline TBD)
  - model readiness

Later, add units to FieldSet metadata following CCPP conventions

[Interfaces likely to keep evolving as needed]

Resources

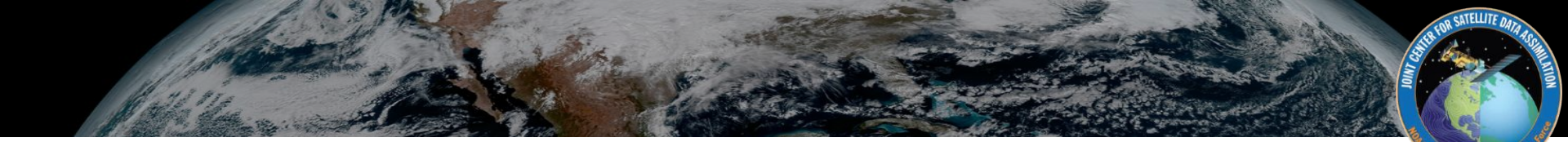
References

- google doc with interface requirements
- these slides, esp. Mesh cartoons in final section

Discussions

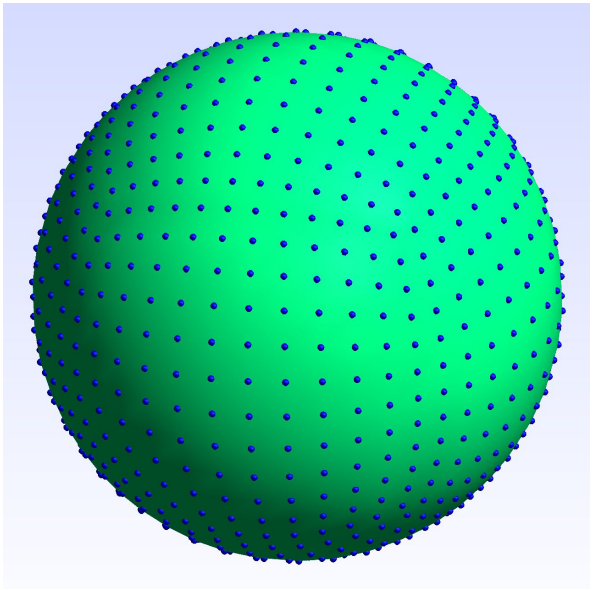
- github discussions page
- drop-in Q&A with JEDI ALGO: Thursdays directly after the JEDI weekly meeting
- we'll schedule additional meetings as needed

Contact me: hebert@ucar.edu

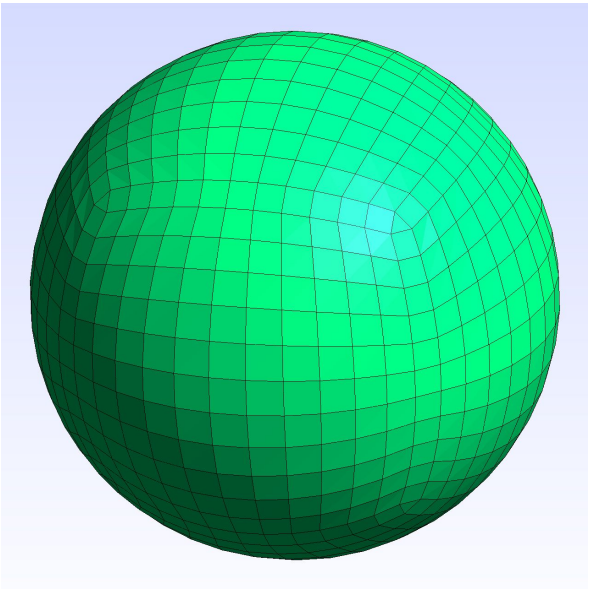


Importing model connectivity into atlas::Mesh

atlas::Mesh



PointCloud encodes no connectivity



FunctionSpace with Mesh enables atlas algorithms like interpolation, halo-building

atlas::Mesh

How to construct atlas::Mesh to match model?

- directly from atlas if grid is available as builtin:
  - latlon grids
  - various gaussian grids
  - some cubed-sphere grids (LFRic, but not FV3)
- import model coordinates + connectivity into atlas via MeshBuilder
- import model coordinates into atlas, triangulate into unstructured grid

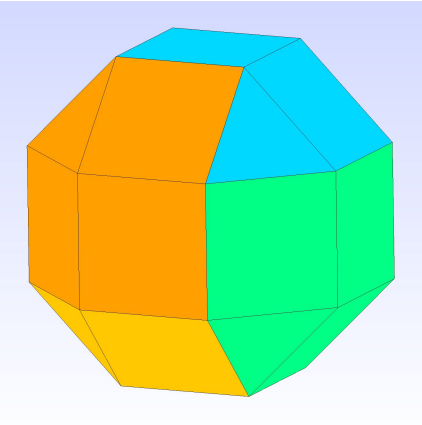
atlas::Mesh

How to construct atlas::Mesh to match model?

- directly from atlas if grid is available as builtin:
  - latlon grids
  - various gaussian grids
  - some cubed-sphere grids (LFRic, but not FV3)
- import model coordinates + connectivity into atlas via MeshBuilder
- import model coordinates into atlas, triangulate into unstructured grid

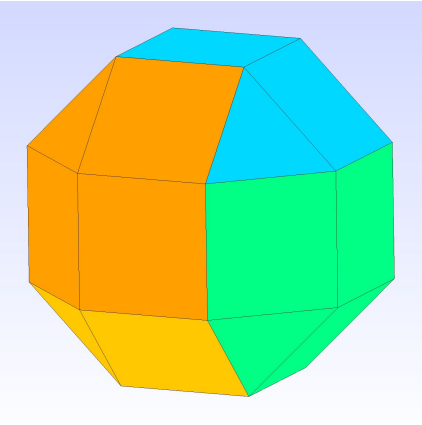
correct for most models, but it's work to prepare the connectivity data

Importing a simple grid with MeshBuilder



Let’s import this C2 cubed-sphere dual into atlas

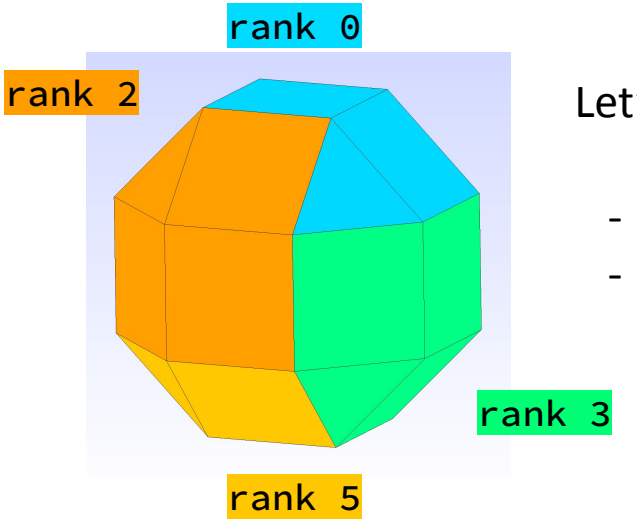
Importing a simple grid with MeshBuilder



Let’s import this C2 cubed-sphere dual into atlas

- connect nodes into cells: triangles or quads

Importing a simple grid with MeshBuilder

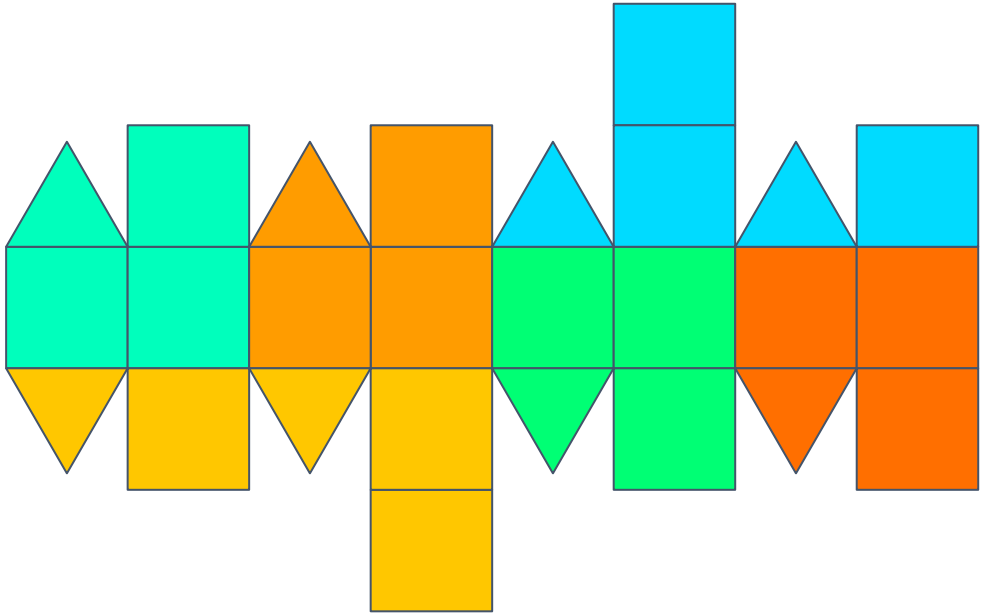


Let’s import this C2 cubed-sphere dual into atlas

- connect nodes into cells: triangles or quads
- assign cells to partitions (MPI ranks)

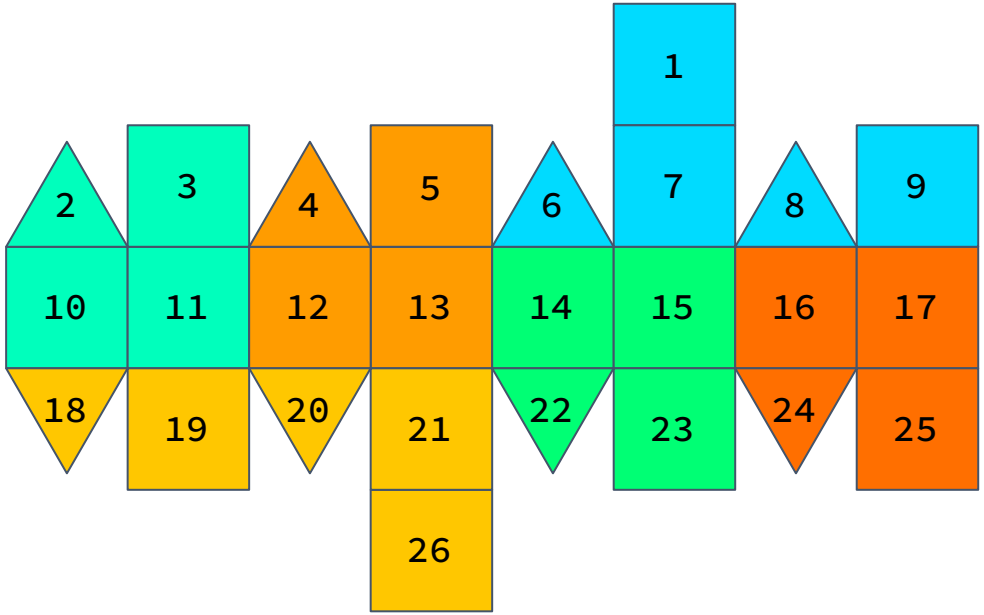
Importing a simple grid with MeshBuilder

Flatten to show global grid



Importing a simple grid with MeshBuilder

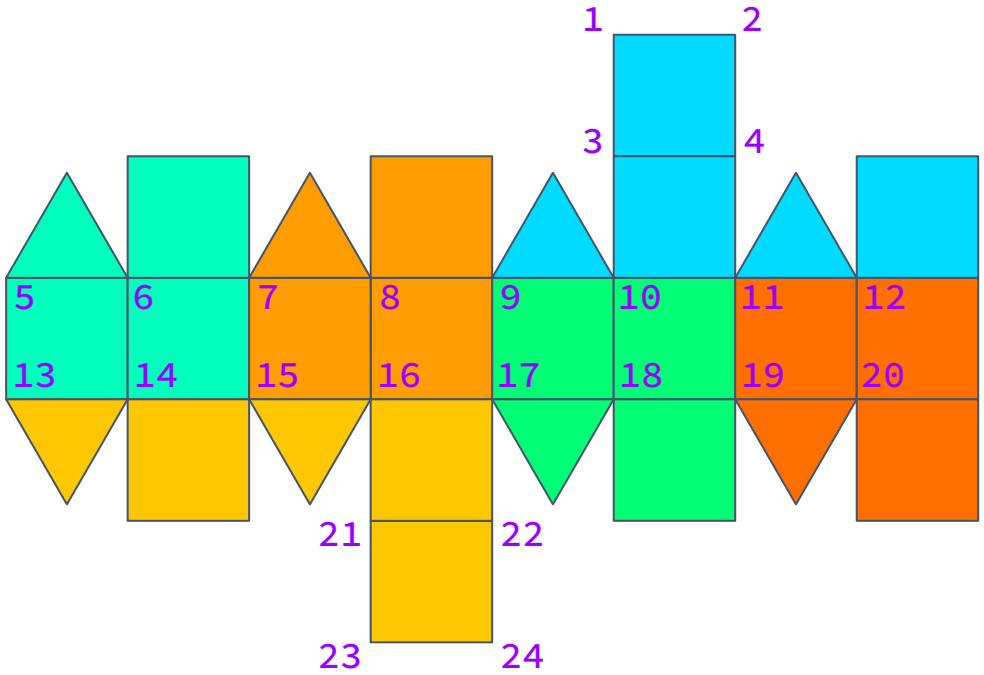
cell global index (1-based):



Importing a simple grid with MeshBuilder

cell global index (1-based):

global index (1-based):



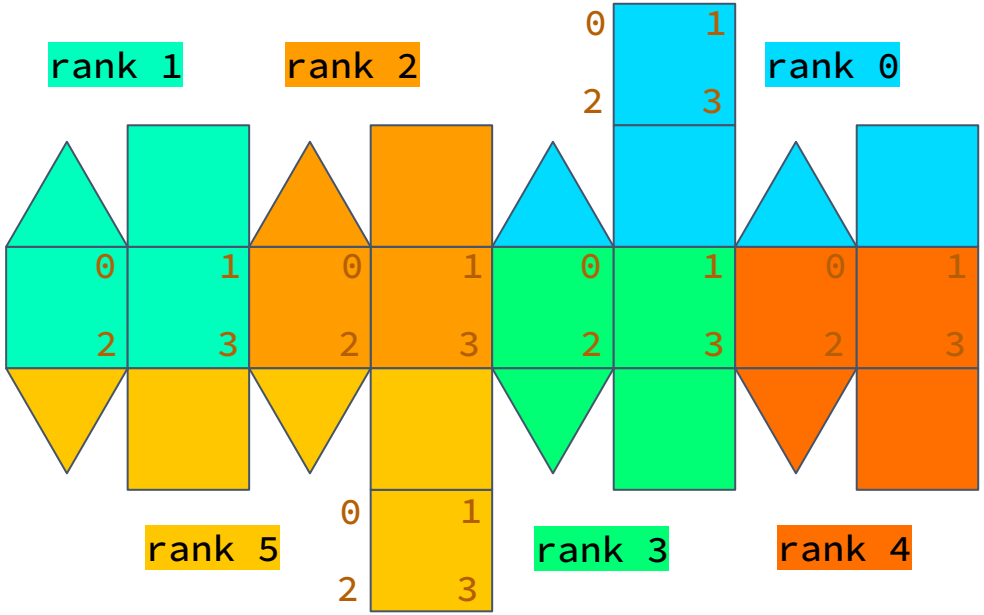
Importing a simple grid with MeshBuilder

cell global index (1-based):

global index (1-based):

partition (MPI rank):

remote index:
local index on owning rank



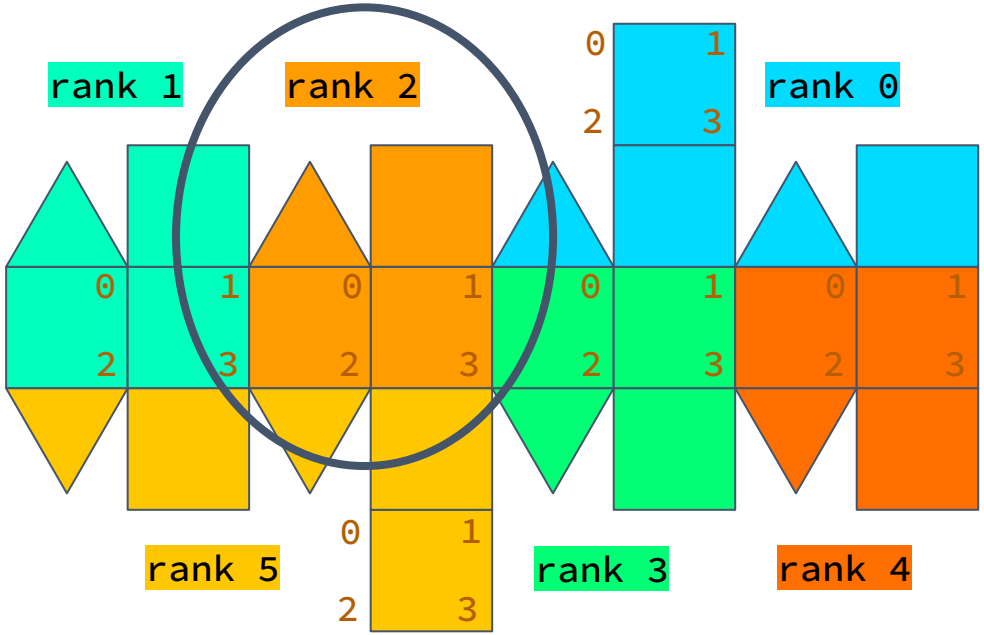
Importing a simple grid with MeshBuilder

cell global index (1-based):

global index (1-based):

partition (MPI rank):

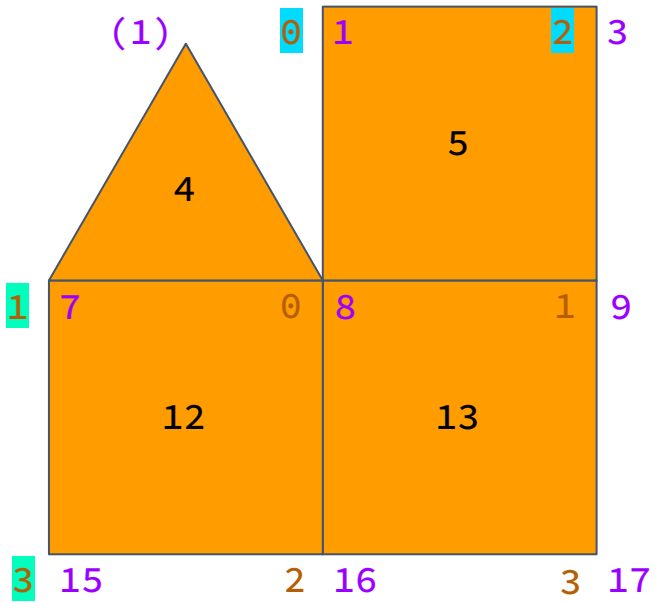
remote index:
local index on owning rank



Importing a simple grid with MeshBuilder

On rank 2, MeshBuilder arguments:

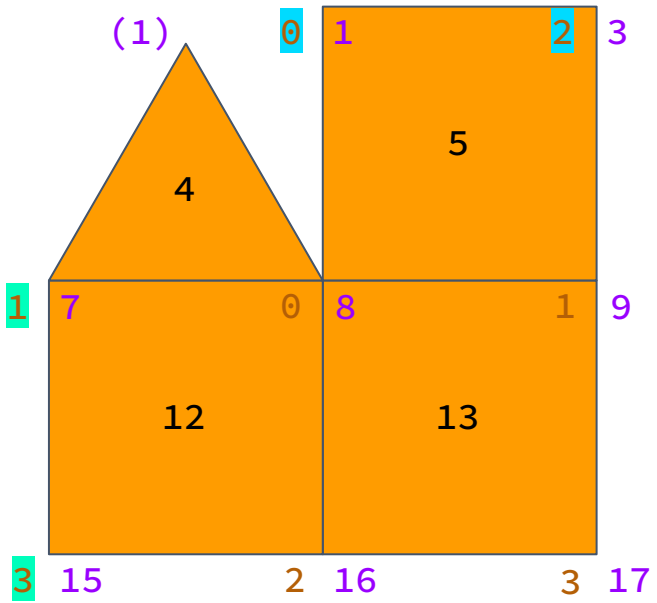
```
num nodes = 8
global indices = {1, 3, 7, 8, 9, 15, 16, 17}
ghosts = {1, 1, 1, 0, 0, 1, 0, 0}
partitions = {0, 0, 1, 2, 2, 1, 2, 2}
remote indices = {0, 2, 1, 0, 1, 3, 2, 3}
```



Importing a simple grid with MeshBuilder

On rank 2, MeshBuilder arguments:

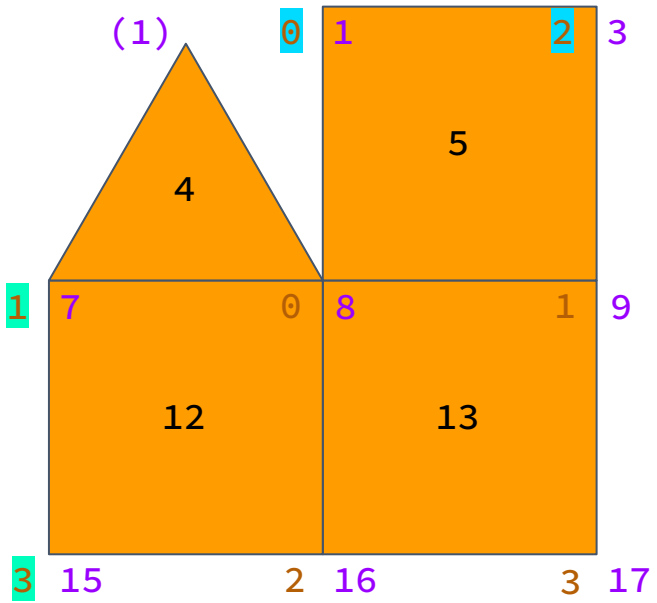
```
num nodes = 8
global indices = {1, 3, 7, 8, 9, 15, 16, 17}
ghosts = {1, 1, 1, 0, 0, 1, 0, 0}
partitions = {0, 0, 1, 2, 2, 1, 2, 2}
remote indices = {0, 2, 1, 0, 1, 3, 2, 3}

num triangle cells = 1
tri cells = {4}
tri bdry nodes = {{1, 7, 8}}
```



Importing a simple grid with MeshBuilder

On rank 2, MeshBuilder arguments:

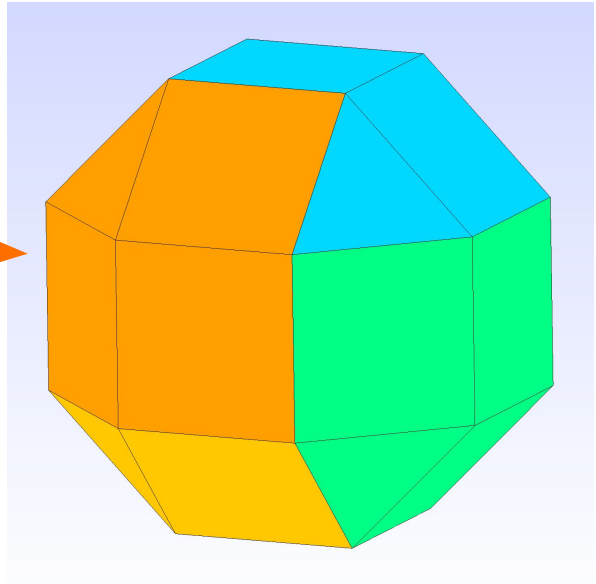
```
num nodes = 8
global indices = {1, 3, 7, 8, 9, 15, 16, 17}
ghosts = {1, 1, 1, 0, 0, 1, 0, 0}
partitions = {0, 0, 1, 2, 2, 1, 2, 2}
remote indices = {0, 2, 1, 0, 1, 3, 2, 3}

num triangle cells = 1
tri cells = {4}
tri bdry nodes = {{1, 7, 8}}

num quad cells = 3
quad cells = {5, 12, 13}
quad bdry nodes = {{1, 3, 8, 9},
                   {7, 8, 15, 16},
                   {8, 9, 16, 17}}
```



Importing a simple grid with MeshBuilder

Passing arguments to MeshBuilder produces example C2 Mesh

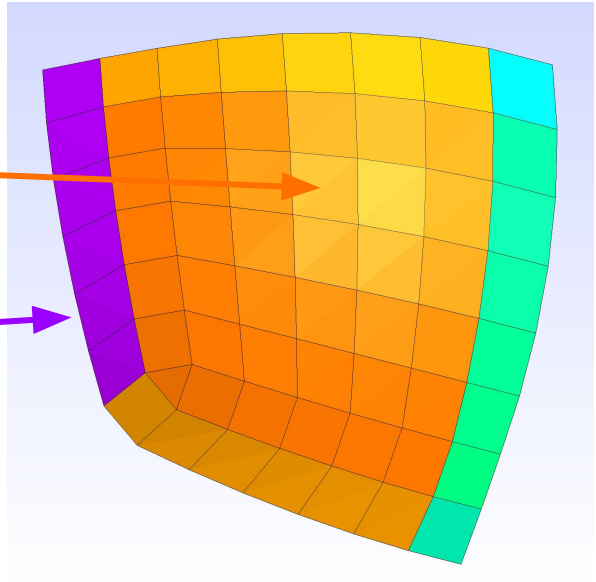
Tile on rank-2 from example



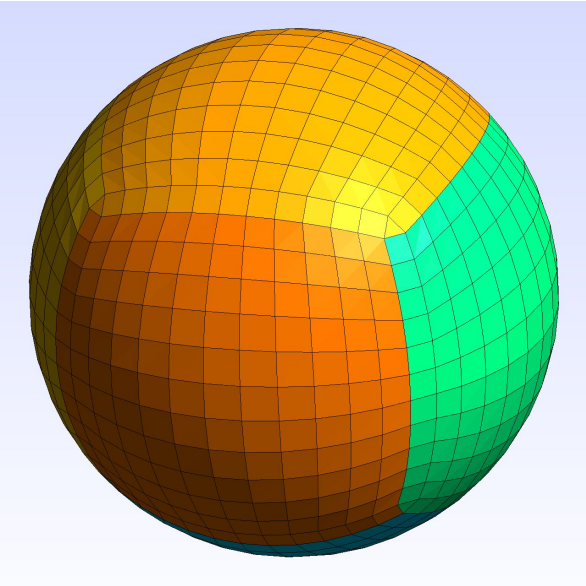
Example with higher-res grid

Tile from C12 grid on 6x2x2 layout

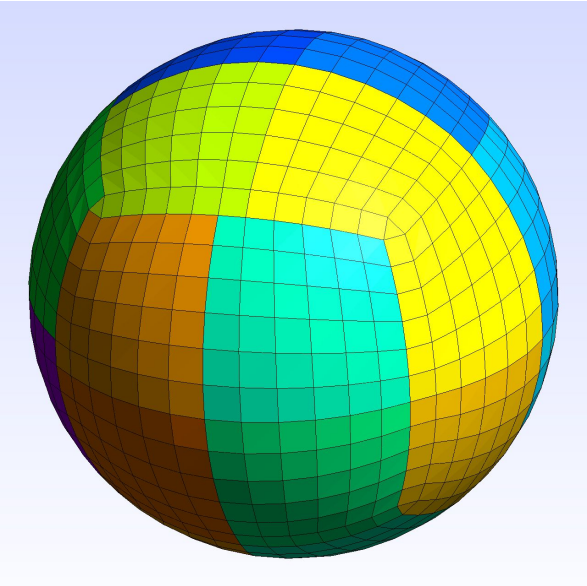
Mesh halo as added by
`atlas::mesh::actions::build_halo`



Example with higher-res grid



C12 grid on 6 tasks



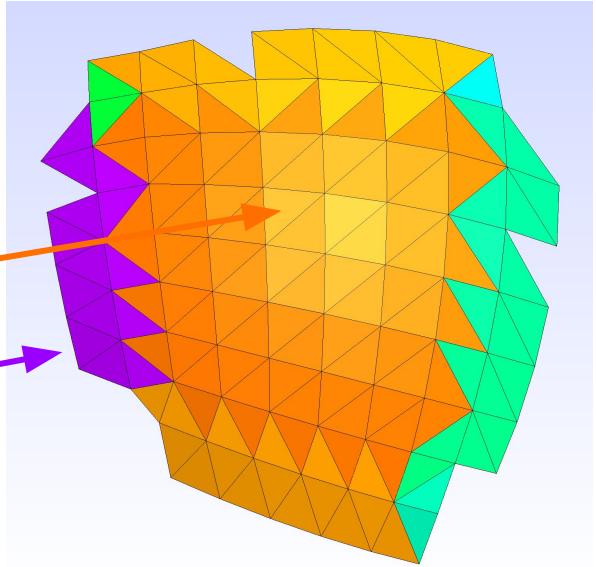
C12 grid on 24 tasks

Connectivity from atlas

Alternatively, let atlas compute the connectivity

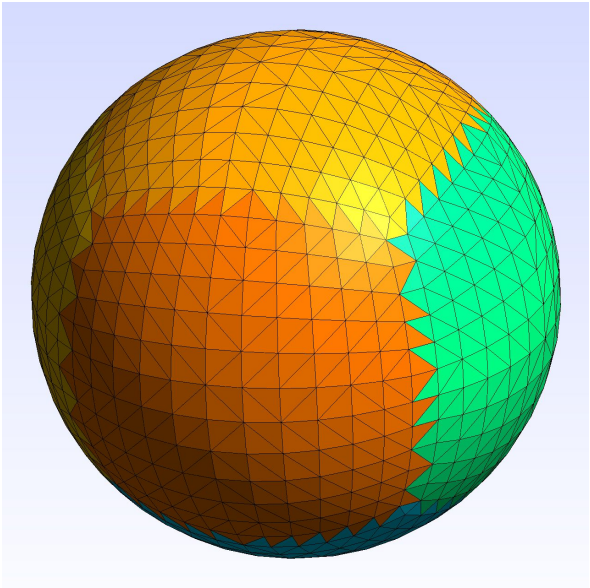
On each processor, gather global list of
- lats, lons, partitions

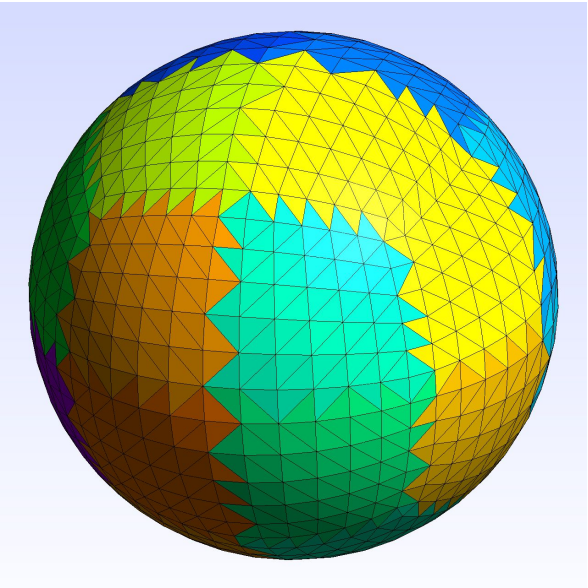
atlas can compute a Delaunay triangulation over the global grid, then trim back down to owned points:



`atlas::mesh::actions::build_halo`

Connectivity from atlas

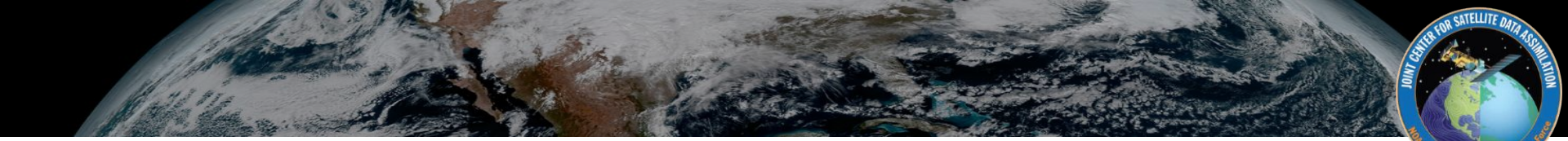




atlas::Mesh

Mesh recap:

- directly from atlas
  - great option if possible
- import model grid connectivity
  - flexible and general
  - must set up the connectivity data to pass to atlas
- import model coordinates, allow atlas to triangulate
  - easy to set up
  - imperfect match to model grid
  - global triangulation process can be slow and MPI-dependent
  - not recommended for operational models



Questions?

Bonus: visualizing meshes

Using Gmsh for mesh visualization:

- Gmsh webpage [version 4.6 recommended, some bugs in more recent versions]
- atlas can output mesh in Gmsh format:

```
#include "atlas/output/Gmsh.h"
// work to construct the mesh
atlas::output::Gmsh gmsh("out.msh",
    atlas::util::Config("coordinates", "xyz")
    | atlas::util::Config("ghost", true));  // enables viewing halos per task
gmsh.write(mesh);
```

To make figures like those in this presentation, follow steps similar to:

- open “out.msh”; on macOS use `` `/Applications/Gmsh.app/Contents/MacOS/gmsh path/to/out.msh` ``
- in Tools > Options > Mesh > Visibility, turn on “2D element faces”, in Tools > Options > Mesh > Color, set coloring mode to “By mesh partition” to color patches by MPI task
- to view mesh (and halo) of N’th MPI task, open “out.mesh.pN” instead

This bonus slide added 8/14/23 after presentation feedback