THE MODEL FOR PREDICTION ACROSS SCALES: MESHES AND SOFTWARE FRAMEWORK

Michael Duda[1],
Todd Ringler[2], and Bill Skamarock[1]

[1]National Center for Atmospheric Research*, NESL
[2]Los Alamos National Lab, COSIM

*NCAR is funded by the National Science Foundation

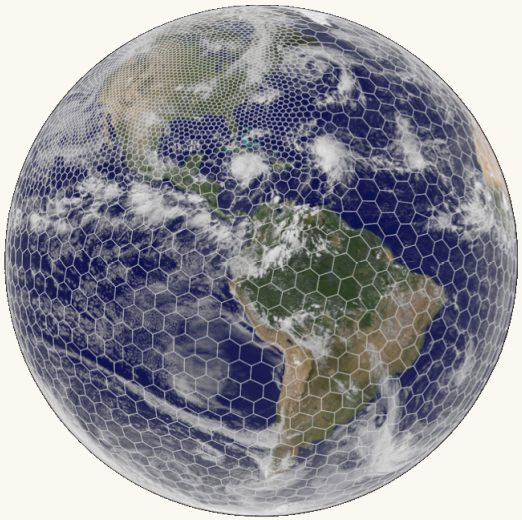
Coupling Technologies for Earth System Modeling
CERFACS Toulouse, France 15-17 December 2010

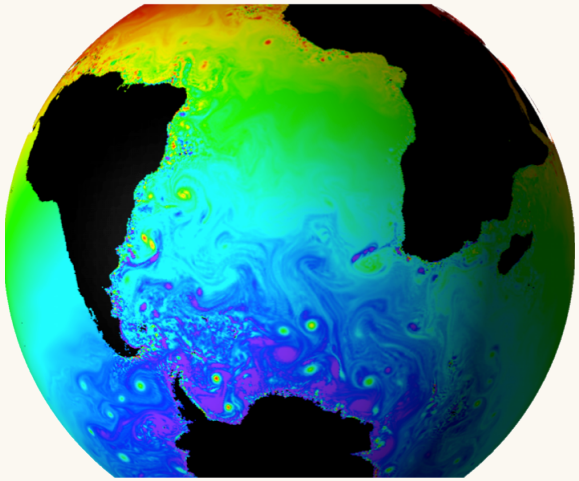
WHAT IS MPAS?

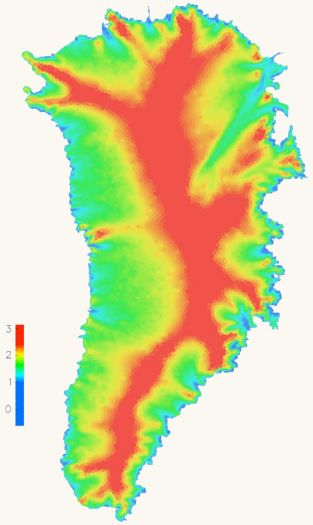
A collaboration between NCAR (MMM) and LANL (COSIM) to develop models for climate, regional climate, and NWP applications:

- MPAS-Atmosphere (NCAR)
- MPAS-Ocean (LANL)
- MPAS-Ice (LANL)
- MPAS framework, infrastructure (NCAR, LANL)

MPAS models are based on the centroidal Voronoi tessellation (CVT) with a C-grid staggering



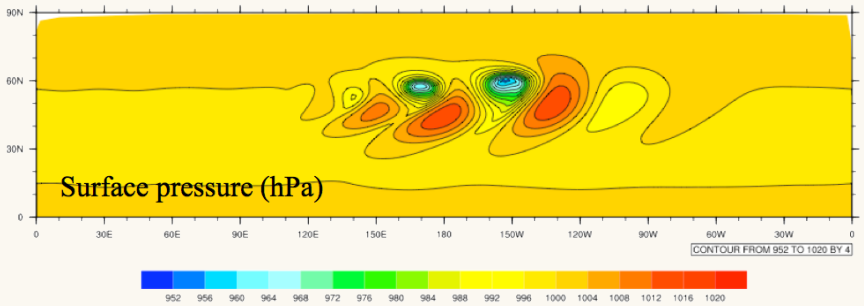




from Ringler et al. (2008)

MPAS FEATURES AND DEVELOPMENT

- Code development began in late 2008
  - Mesh generation, simple toy framework, shallow water model; then hydrostatic atmosphere, ocean, and non-hydrostatic atmosphere
- Atmosphere: Simulate over whole sphere or doubly-periodic planar domains
- Ocean: Simulate over basin (non-simply-connected) domains with boundaries
- Models are conservative (total energy, mass, and potential vorticity)
- Current status
  - Real bathymetry ocean simulations
  - On the verge of real-data simulations in atmosphere cores (idealized cases so far)



Surface pressure field at day 9 from the Jablonowski and Williamson baroclinic wave test case in the MPAS non-hydrostatic core

MPAS INFRASTRUCTURE FOR SUPPORTING CVTs

Development of MPAS models within a common framework is justified by the need for software infrastructure to support modeling on unstructured CVT meshes

- Though MPAS meshes are very general, well-designed infrastructure can support full variety of meshes we might use for MPAS models
- The equations sets for MPAS models are different, but all use common building blocks
  - Models can share diffusion, transport, differential operators, etc. if these are designed in a general way
- Improvements to one model may be leveraged across all MPAS models
- **Working out the details of coupling one model should lead to the possibility of coupling any of the MPAS models**

OUTLINE

1 – The unstructured meshes used in MPAS present interesting opportunities as well as unique challenges for model software and coupling; consequently, I'll discuss CVTs first

From the standpoint of coupler, the meshes on which fields are discretized may be of more interest than the model itself?

2 – Keeping in mind the requirements imposed by meshes, I'll outline the MPAS software framework and summarize how we see coupling working in MPAS

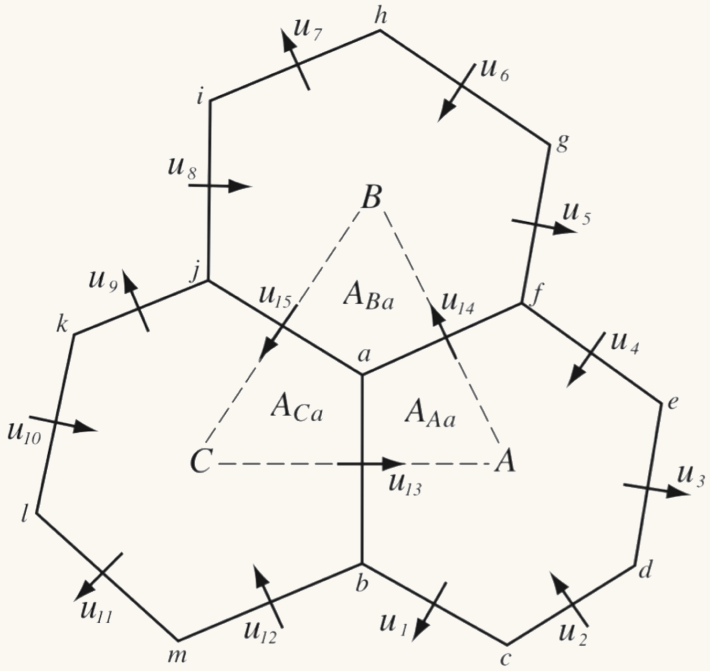
from G. Barlow, Hexagonal territories, Animal Behav., 22 (1974), pp. 876–878.

MPAS MESHES

The defining feature of MPAS models is their use of centroidal Voronoi tessellations with a C-grid staggering

Voronoi := each grid volume (cell) V_i is associated with a unique *generating point* $\boldsymbol{x}_i$ such that all points within V_i are closer to $\boldsymbol{x}_i$ than to any other $\boldsymbol{x}_j$

Centroidal := the generating point for each Voronoi cell is also the mass centroid of that cell (w.r.t. some density function)

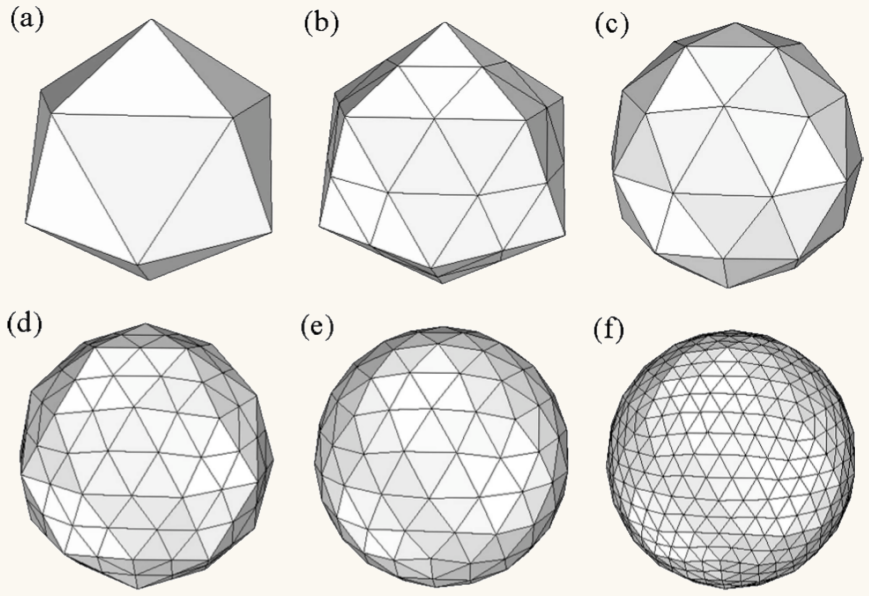


Prognostic velocities are velocities normal to cell faces ("edges") at the point where the edge intersects the arc joining cells on either side

GENERATING SCVTs

For quasi-uniform SCVTs, we can employ successive subdivision of the icosahedron

- The vertices of these triangular meshes can be used as the generating points for a spherical Voronoi tessellation
- To create a spherical ***centroidal*** Voronoi tessellation, some iteration is required

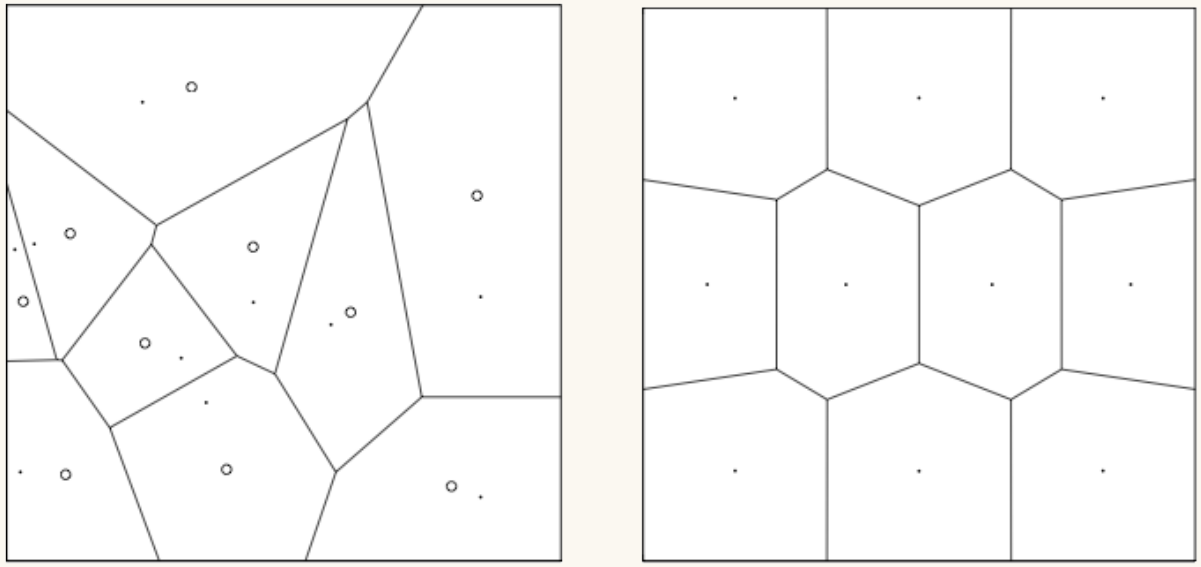


From Lipscomb and Ringler (2005)

ENTER LLOYD'S METHOD

Given an initial set of generating points, Lloyd's method may be used to arrive at a CVT:

1. Begin with any set of initial points (the generating point set)
2. Construct a Voronoi diagram for the set
3. Locate the mass centroid of each Voronoi cell
4. Move each generating point to the mass centroid of its Voronoi cell
5. Repeat 2-4 to convergence



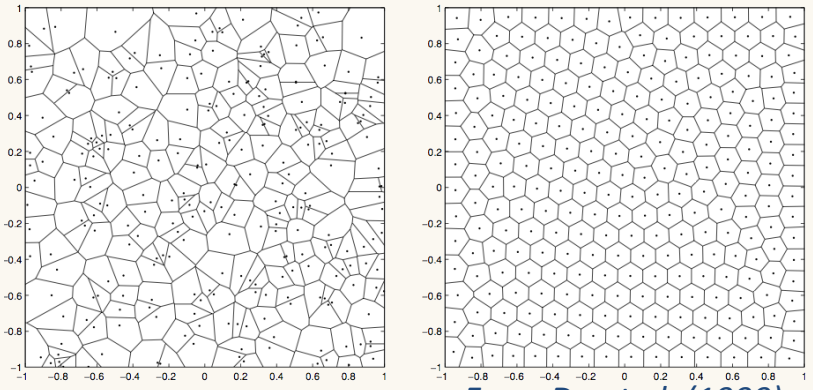
From Du et al. (1999)

MacQueen's method, an randomized alternative to Lloyd's method may also be used; no Voronoi diagrams need to be constructed, but convergence is generally much slower

DENSITY FUNCTION – KEY TO REFINEMENT

Lloyd's method can be viewed as the minimization of an energy functional; in the plane, it can be shown that hexagonal Voronoi cells provide the minimum energy configuration *for constant density*

To create regions of grid refinement, we simply define a non-uniform density function over the domain, and use this when computing the mass centroids of Voronoi cells in Lloyd's method



From Du et al. (1999)

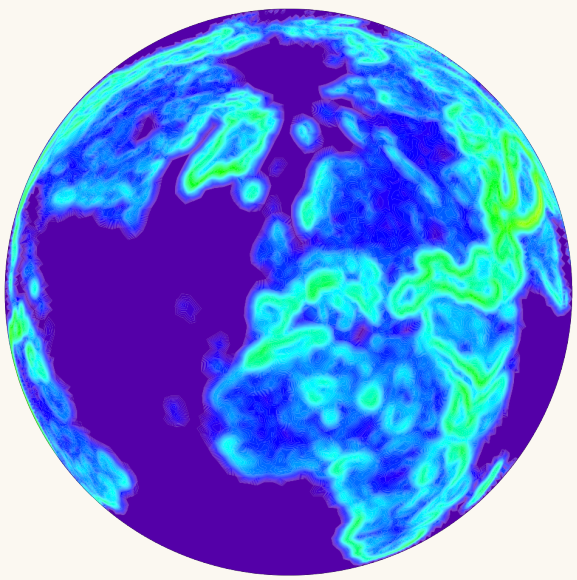
For a density function $\rho(\boldsymbol{x}) > 0$, it is conjectured (Ju et al. (2010)) that, as the number of Voronoi cells increases, the diameters, h_i and h_j, of Voronoi cells associated with generating points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are related by

$$\frac{h_i}{h_j} \approx \left(\frac{\rho(\mathbf{x}_j)}{\rho(\mathbf{x}_i)}\right)^{1/(d'+2)}$$

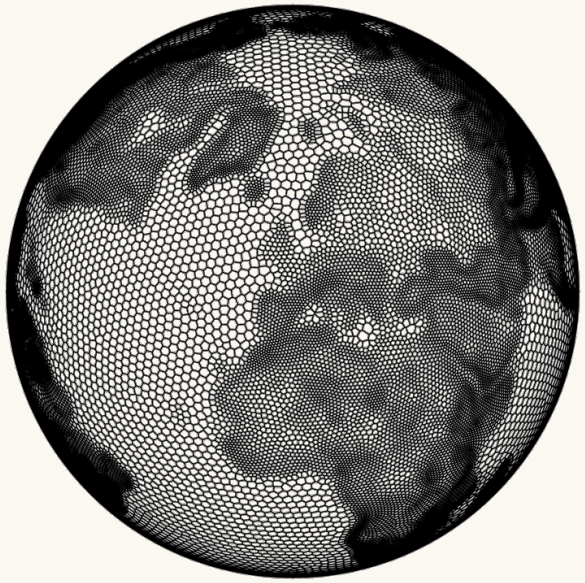
with $d' = 2$.

AN ABSURD EXAMPLE OF MESH REFINEMENT

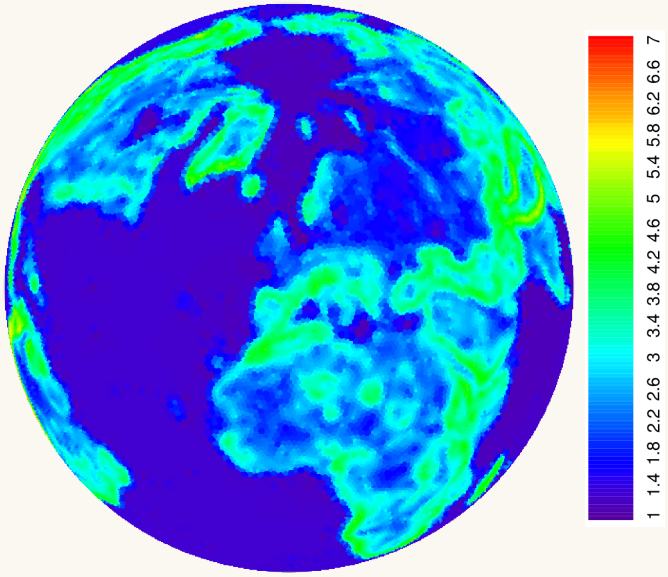
Define ρ(***x***) as a function of the magnitude of the topography gradient, for example:



Fourth root of density function



Resulting SCVT



Normalized inverse mean distance of cell centroid from each of its neighboring centroids

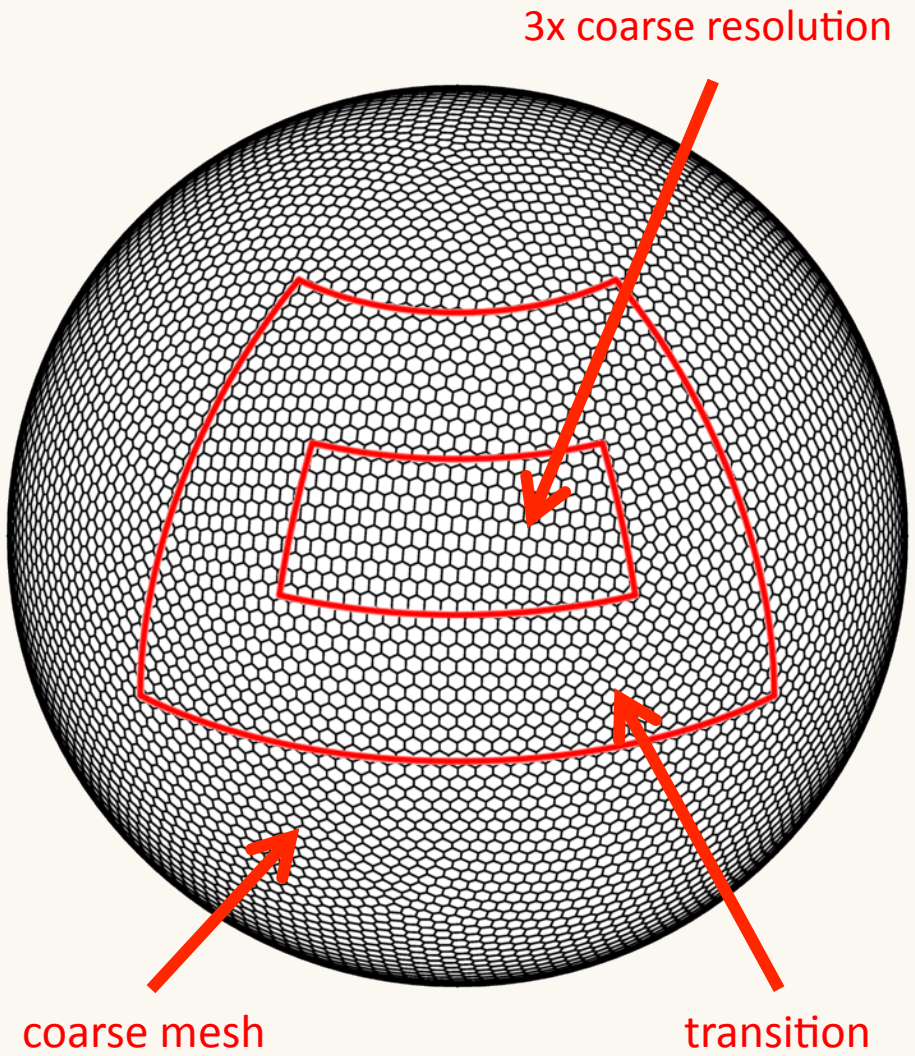
Theoretically, this is a valid MPAS mesh, though evidence from initial testing of the MPAS non-hydrostatic atmosphere code on multi-resolution meshes on the Cartesian plane suggests that smoother refinement (i.e., less abrupt transitions) produces better solutions

POSSIBLE WAYS TO ARRIVE AT LOCAL REFINEMENT

A simple-minded approach to generating a variable-resolution mesh:

1. Establish desired relative resolutions over simulation domain
2. Define density function as the fourth power of relative resolution
3. Beginning with a uniform mesh with appropriate number of cells, run Lloyd's method to convergence

The drawback of this approach is that the absolute resolution in any area of the mesh is difficult to accurately predict

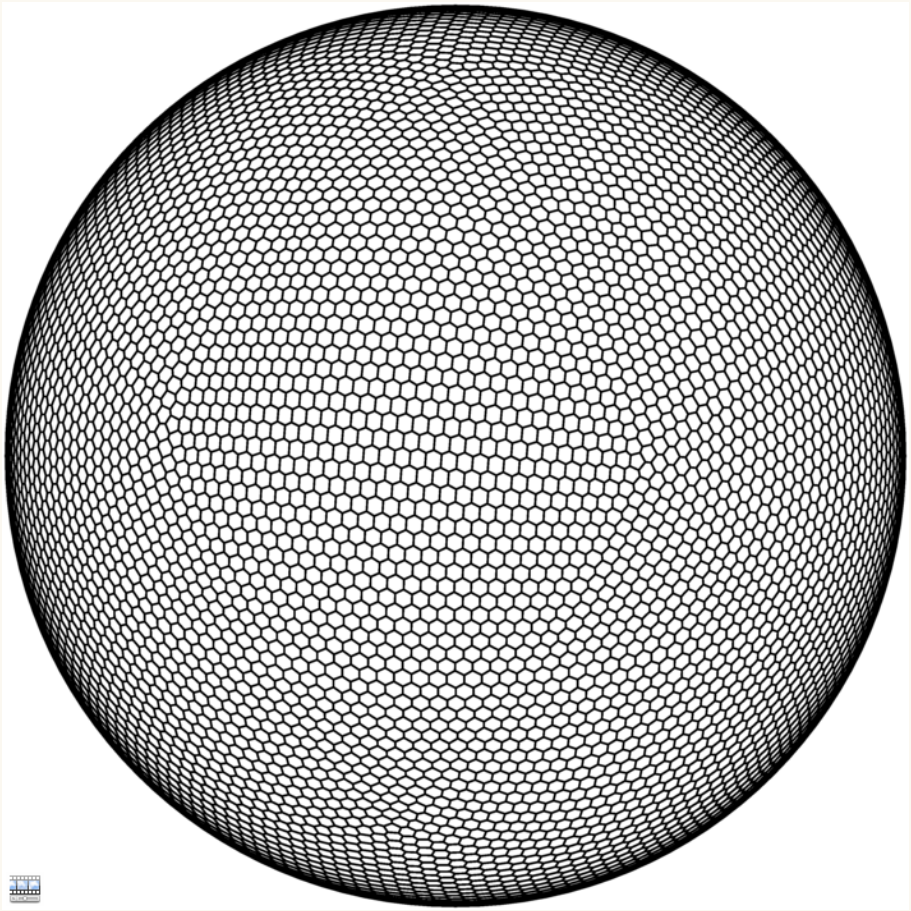


POSSIBLE WAYS TO ARRIVE AT LOCAL REFINEMENT

A simple-minded approach to generating a variable-resolution mesh:

1. Establish desired relative resolutions over simulation domain
2. Define density function as the fourth power of relative resolution
3. Beginning with a uniform mesh with appropriate number of cells, run Lloyd's method to convergence

The drawback of this approach is that the absolute resolution in any area of the mesh is difficult to accurately predict

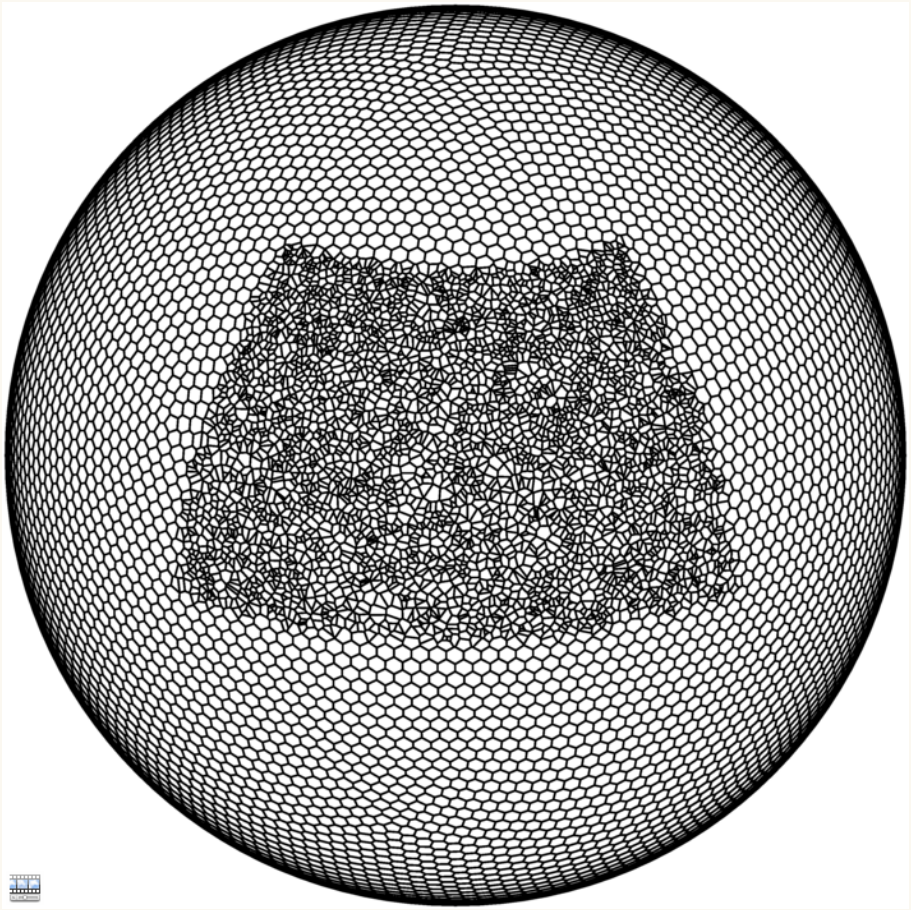


For animation, please see
http://www.mmm.ucar.edu/people/duda/files/mpas/talks/voronoi1.avi

POSSIBLE WAYS TO ARRIVE AT LOCAL REFINEMENT

An alternative strategy that maintains the original coarse-mesh resolution is to add refinement points in the region of refinement

- Hold cells in coarse region fixed while iterating only in the refinement region(s)
- Most applicable where there are regions of constant density with transitions between density regions

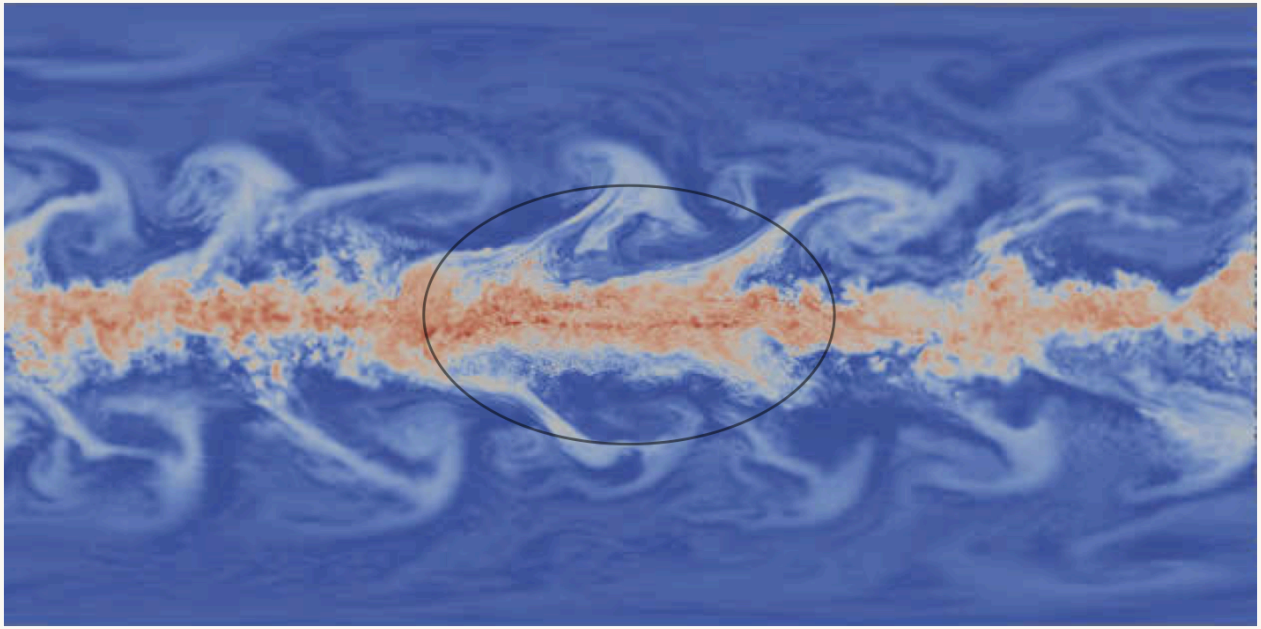


For animation, please see
http://www.mmm.ucar.edu/people/duda/files/mpas/talks/voronoi2.avi

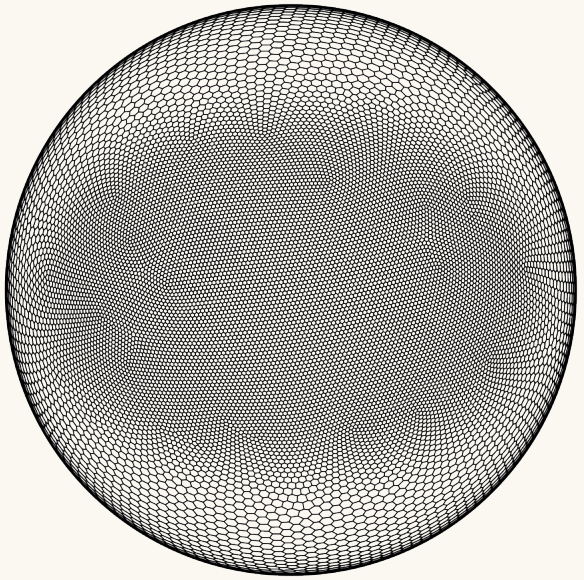
LOCAL REFINEMENT IN ACTION

Results from a multi-resolution full-physics aqua planet simulation (MPAS hydrostatic atmosphere within CCSM CAM4) conducted by LLNL colleagues

- A few details about implementation of hydrostatic core in CAM later…

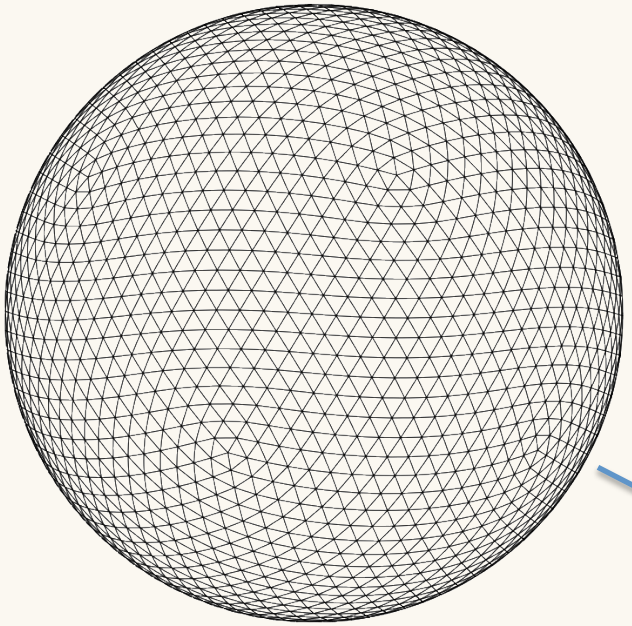


Water vapor, level 5, end of month 15; ellipse designates region of refinement (coarse resolution 120-km, fine resolution 40 km)



Coarser version of meshed used in multi-resolution aqua planet simulation at left

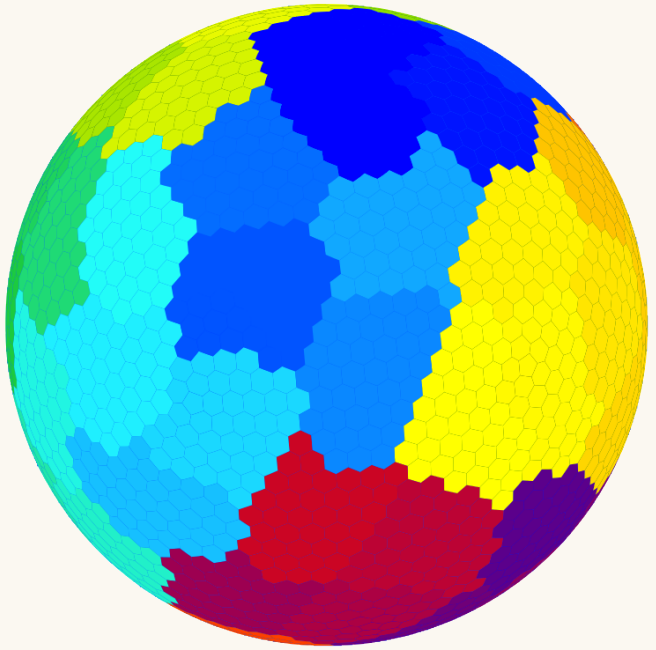
PARALLEL DECOMPOSITION



The *dual* mesh of a Voronoi tessellation is a Delaunay triangulation – essentially the connectivity graph of the cells

Parallel decomposition of an MPAS mesh then becomes a graph partitioning problem*:* ***equally distribute nodes among partitions (give each process equal work) while minimizing the edge cut (minimizing parallel communication)***

Graph partitioning



We use the Metis package for parallel graph decomposition

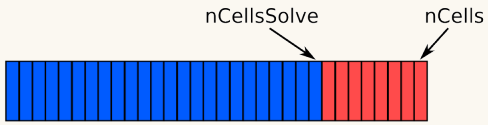
- Currently done as a pre-processing step, but could be done "on-line"

Metis also handles weighted graph partitioning

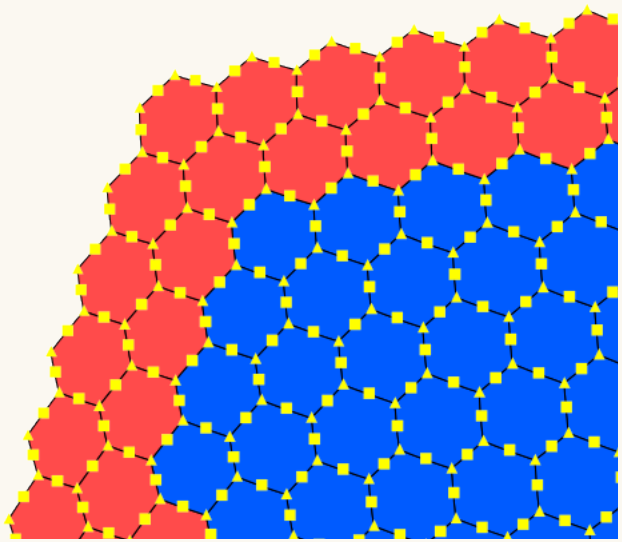
- Given *a priori* estimates for the computational costs of each grid cell, we can better balance the load among processes

PARALLEL DECOMPOSITION (2)

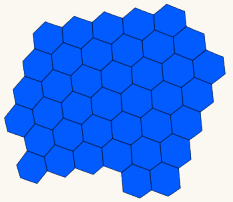
Given an assignment of cells to a process, any number of layers of halo (ghost) cells may be added



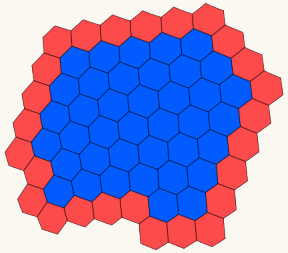
Cells are stored in a 1d array (2d with vertical dimension, etc.), with halo cells at the end of the array; the order of real cells may be updated to provide better cache re-use



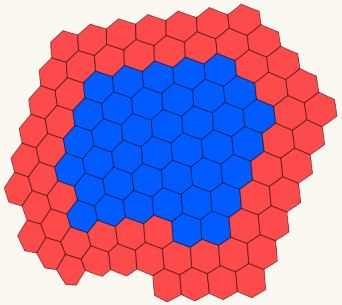
With a complete list of cells stored in a block, adjacent edge and vertex locations can be found; we apply a simple rule to determine ownership of edges and vertices adjacent to real cells in different blocks



Block of cells owned by a process



Block plus one layer of halo/ghost cells



Block plus two layers of halo/ghost cells

CELL REORDERING

- Experiments applying HSFC ordering in MPAS
  - Reorder cells (ZC); reorder cells and edges (ZCE); reorder cells, edges, and vertices (ZCEV)
- Table shows % improvement in runtime over original ordering (positive numbers designate improvement)
  - 4 different problems/grid sizes
  - 16 processor runs

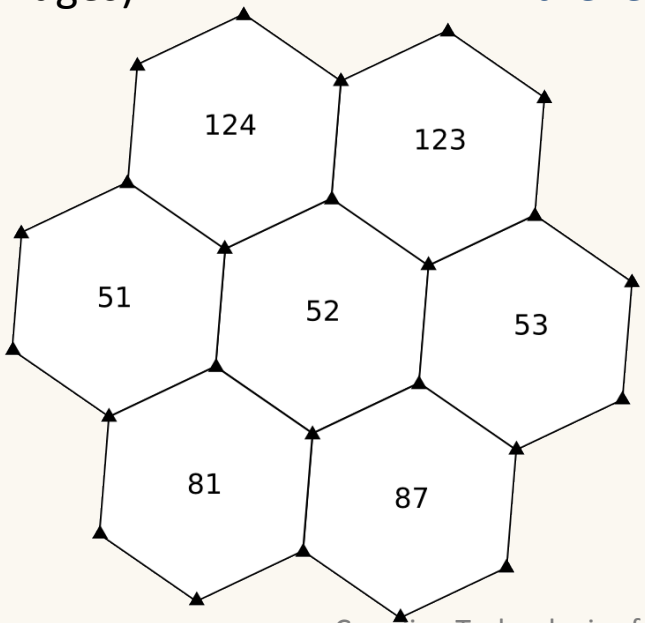
Grid size	ZC	ZCE	ZCEV
40962	-0.1%	-1.0%	1.1%
163842	13.4%	14.8%	15.6%
655362	19.0%	19.4%	17.7%
2621442	23.0%	24.9%	20.6%

From Michael Wolf and Karen Devine (Sandia)

MESH REPRESENTATION

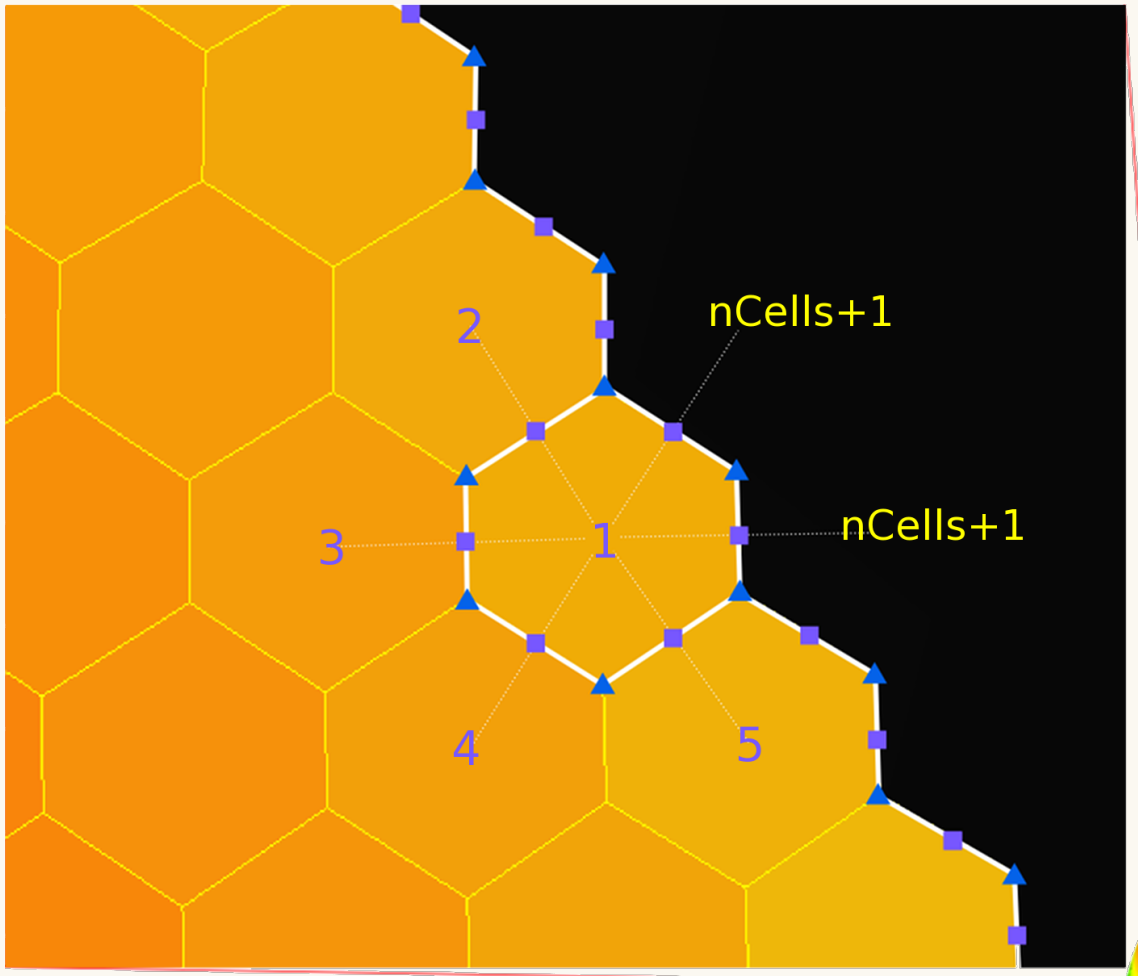
Meshes are represented in MPAS by a set of indexing and geometry arrays:

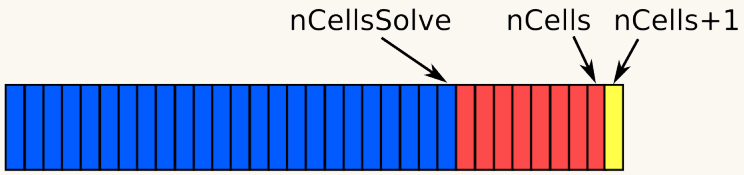
cellsOnCell(maxEdges, nCells)	– the indices of cells adjacent to a cell
edgesOnCell(maxEdges, nCells)	– the indices of edges of a cell
verticesOnCell(maxEdges, nCells)	– the indices of vertices (corners) of a cell
nEdgesOnCell(nCells)	– the number of edges of a cell
edgesOnVertex(3, nVertices)	– the indices of edges incident with a vertex
areaCell(nCells)	– the area of a cell
dvEdge(nEdges)	– the length (vertex-to-vertex) of an edge
…	




```
nEdgesOnCell(52) = 6
cellsOnCell(1,52) = 51
cellsOnCell(2,52) = 81
cellsOnCell(3,52) = 87
cellsOnCell(4,52) = 53
cellsOnCell(5,52) = 123
cellsOnCell(6,52) = 124
```

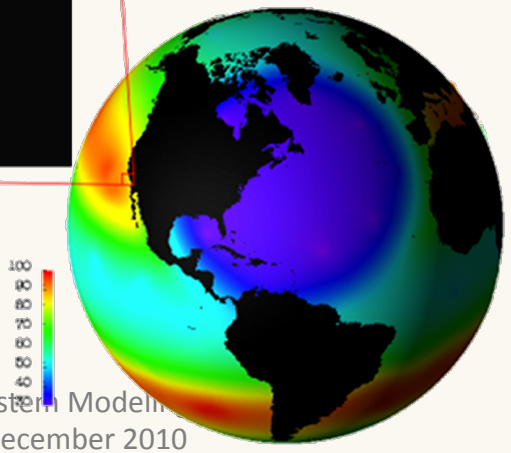
BOUNDARIES IN MPAS MESHES





After generating a mesh, cells are removed from the mesh over "invalid" regions

- Boundary is composed of a set of cell edges and vertices
- **cellsOnCell** values referencing non-existent cells are set to a "garbage" cell
- Boundary edges and vertices marked; can enforce no-flux
- Computation of fields on boundaries (e.g., vorticity at boundary vertices) can use normal loops if proper values are assigned to "garbage" cell/edge/vertex



REGIONS OF DIFFERENT TIME STEP

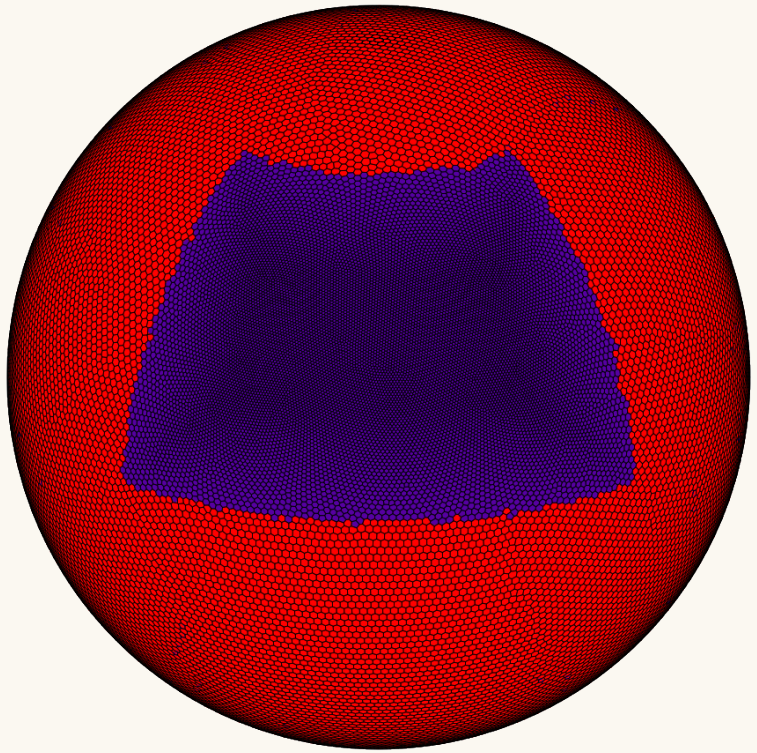
For compact regions of refinement:

- We must use a smaller time step in the refined region
- But we don't want to limit time step unnecessarily in coarse regions

- Solution: Use different time steps in different parts of the grid

Example: Blue cells represent ~31% of total number of cells

- For a 3:1 refinement in blue region, using a global timestep 1/3 of that required for coarse mesh would require 85% more CPU time than using a 1/3 dt timestep in the blue region only
- Added cost of using a single global timestep decreases as refinement ratio increases (for fixed refinement area)
- Differences in physics parameterization costs between regions should also be considered



FLEXIBILITY IN BLOCK DECOMPOSITION

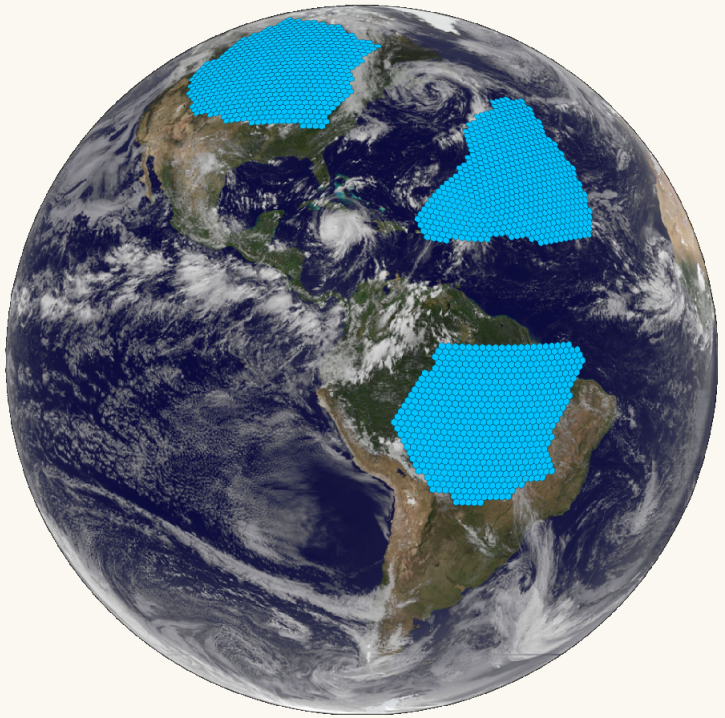
The ability to partition a mesh into more blocks than there are processors may prove useful

- Although current prototype MPAS infrastructure doesn't handle multiple blocks per process, we intend to support this in future

Load balancing:

Assign a process blocks from different parts of the domain to minimize load imbalance

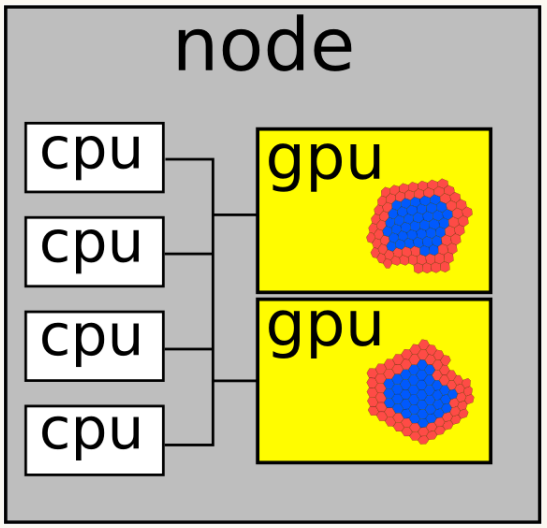
- From opposite sides of the earth to balance "day" and "night" computation
- From tropical, mid-, and high-latitude regions for physics load balancing
- Algorithms to assign blocks from unstructured meshes could be interesting!

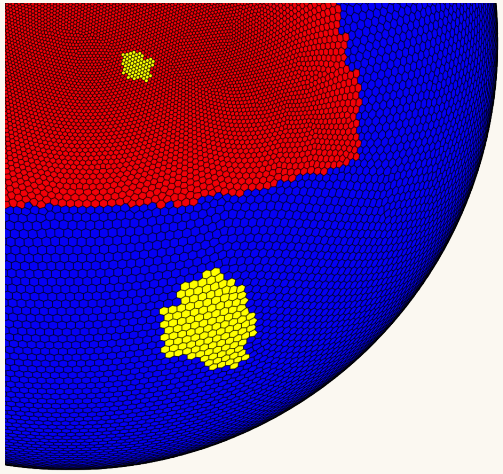


FLEXIBILITY IN BLOCK DECOMPOSITION

Assigning work to accelerator hardware:

On machines with accelerator devices attached to nodes, have one process per node with a number of blocks equal to the number of accelerators





For different time-step regions:

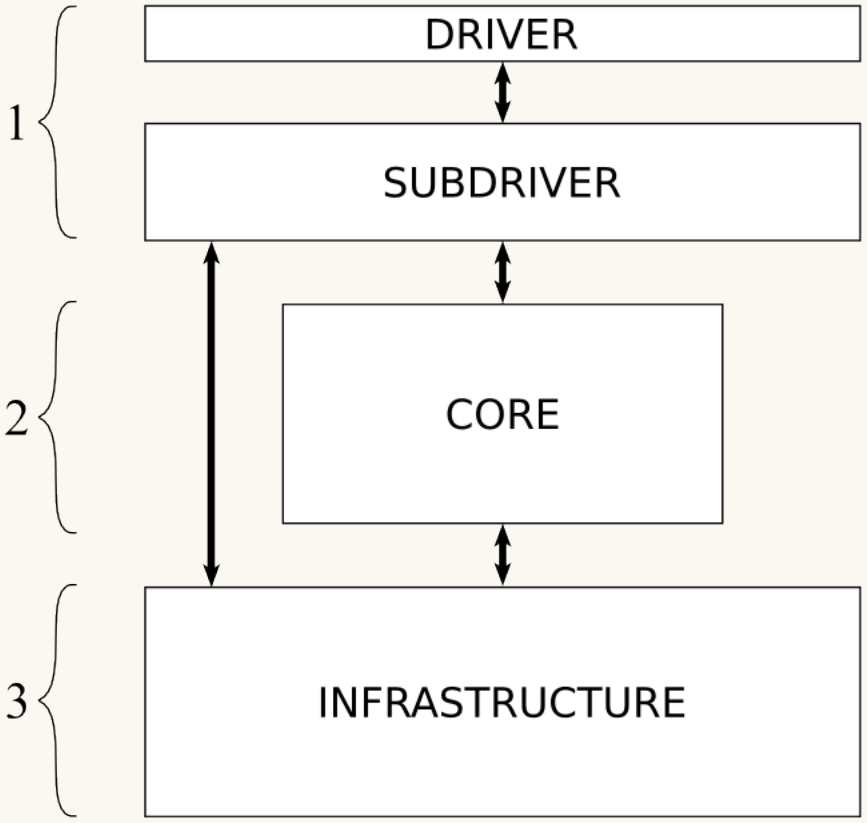
Each process owns one block from each time-step region (i.e., one block from the fine-mesh region and one block from the coarse mesh region)

In general, it would be nice if couplers supported disconnected sets of cells on each process; at higher resolutions, this may be a requirement for coupling with ocean models anyway!

MPAS SOFTWARE: SOME BASIC FACTS

- The MPAS software is (still!) our first prototype
  - Infrastructure has been extended, revised in minor ways to meet immediate needs in core development, but no wholesale redesign
  - **The architecture that I'll present is a mix of what we have currently and what we have planned**
- All code in Fortran
  - This may seem like a given, but perhaps the choice of language is not obvious; the current MPAS infrastructure maps nicely onto objects
  - **It would be nice if coupling frameworks weren't tied to a particular language (use ISO_C_BINDING, Babel, etc.?)**
- We're ready to draft a proper architecture for MPAS
  - This would be the obvious time to incorporate aspects necessary for easy coupling in future

MPAS SOFTWARE ARCHITECTURE



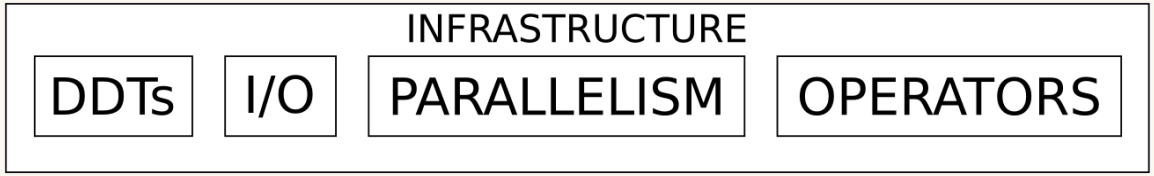
***Arrows** indicate interaction between components of the MPAS architecture*

1. **Driver layer** – The high-level DRIVER and SUBDRIVER are ignorant of the details of the particular MPAS core.

2. **MPAS core** – The MPAS CORE performs the computational work of MPAS, employing data structures and functionality of the INFRASTRUCTURE.

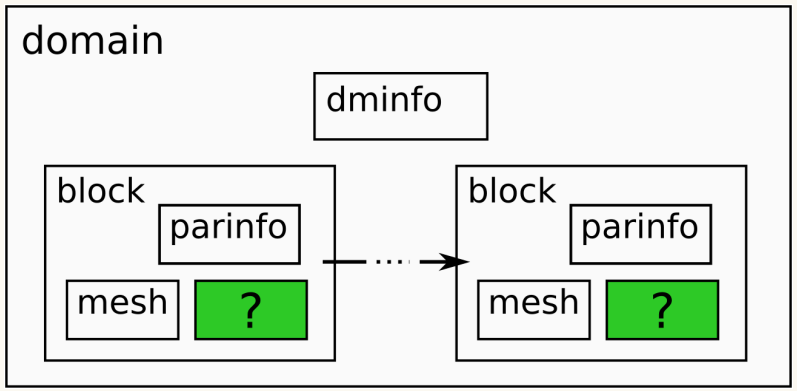
3. **Infrastructure** – The infrastructure provides DDTs that the I/O, parallelism, and operators implemented in the infrastructure work with.

MODEL INFRASTRUCTURE



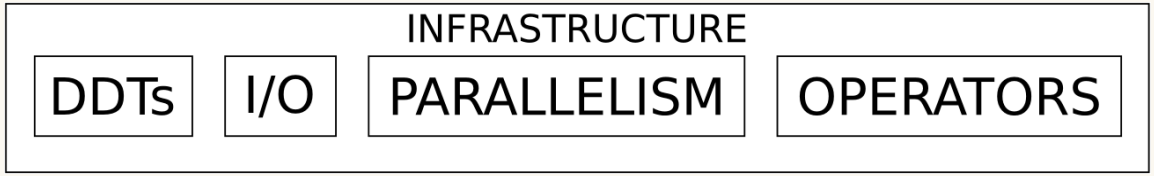
DDTs: The MPAS infrastructure contains definitions for derived data types

- *domain* encapsulates complete state of computational domain for a process
- *block* contains model fields, mesh description, and parallel information for a single block
- *field* stores single field's data and metadata on a block
  - *fields* are packaged in container types (green box) for convenience within a core
  - MPAS model core ultimately uses field array component directly from *field* types
- **Packaging of fields means that infrastructure DDTs are customized for a core**



- *dminfo* contains MPI communicator and other information used by PARALLELISM
- *parinfo* contains information about which cells/edges/vertices in a mesh are in a the halo region, etc.

MODEL INFRASTRUCTURE (2)



I/O: The high-level MPAS I/O interface deals in infrastructure DDTs

- An I/O DDT can naturally encapsulate, e.g., which fields are written to an I/O stream
- Underlying I/O functionality can be provided by another package, e.g., PIO

PARALLELISM: Implements operations on *field* types needed for parallelism

- e.g., add halo cells to a block, halo cell update, scatter/gather
- callable from either serial or parallel code (no-op for serial code)
- for multiple blocks per process, differences between shared-memory and MPI are hidden

OPERATORS: Provides implementations of general operations on CVT meshes

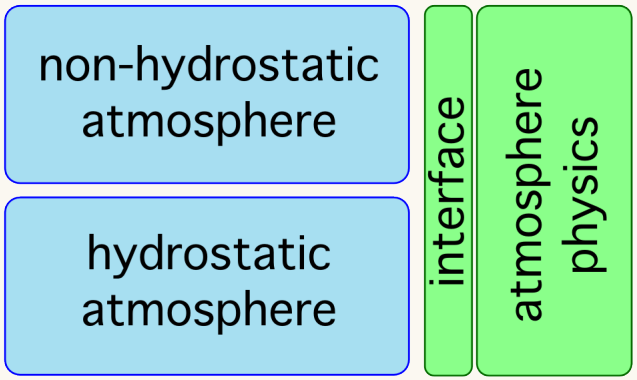
- div and curl operators, interpolation, advection (?), etc.
- these are “building blocks” of MPAS models

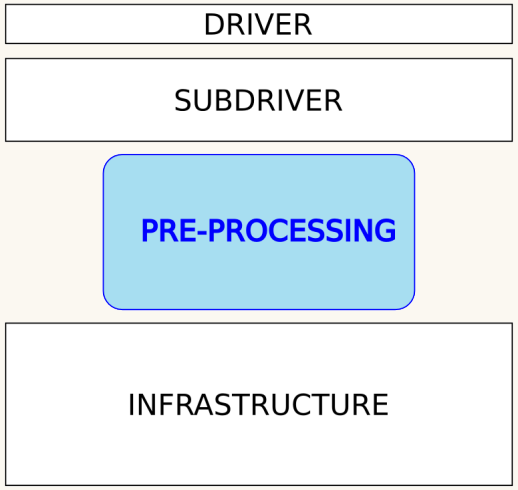
MPAS MODEL CORE

The particular MPAS core to be used is a compile-time decision

For MPAS models, the core is a combination of dynamics and physics

- The core's *run()* routine contains time loop, since time integration is core-specific
- Both atmospheric dynamical cores share the same physics routines
- **The core is passed higher-level DDTs, but use arrays directly from *field* DDTs provided by infrastructure**

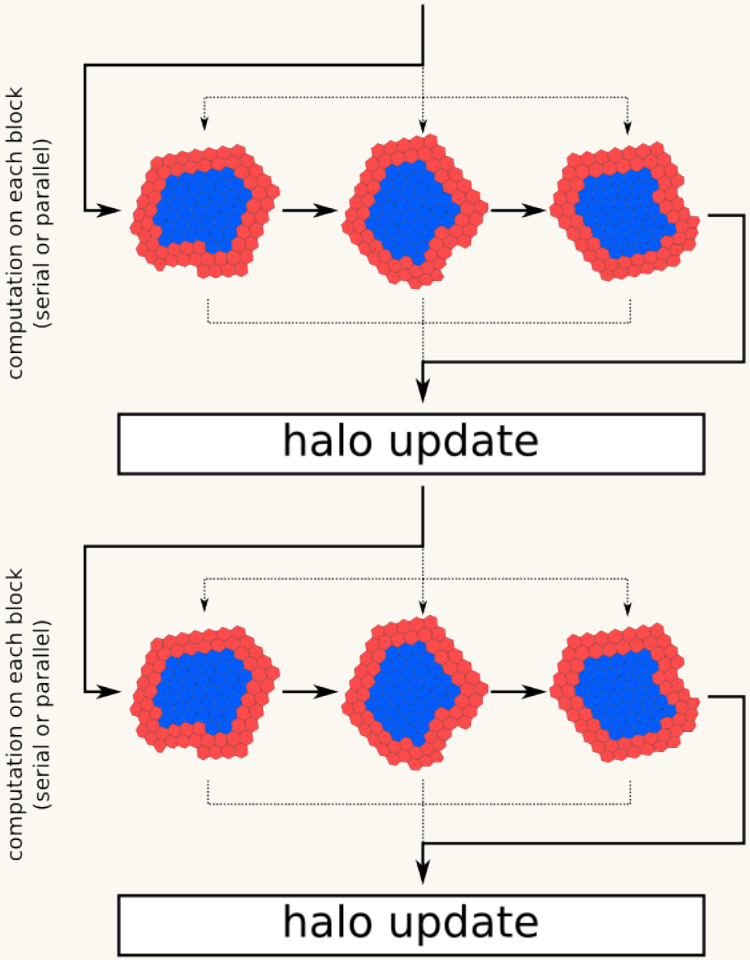




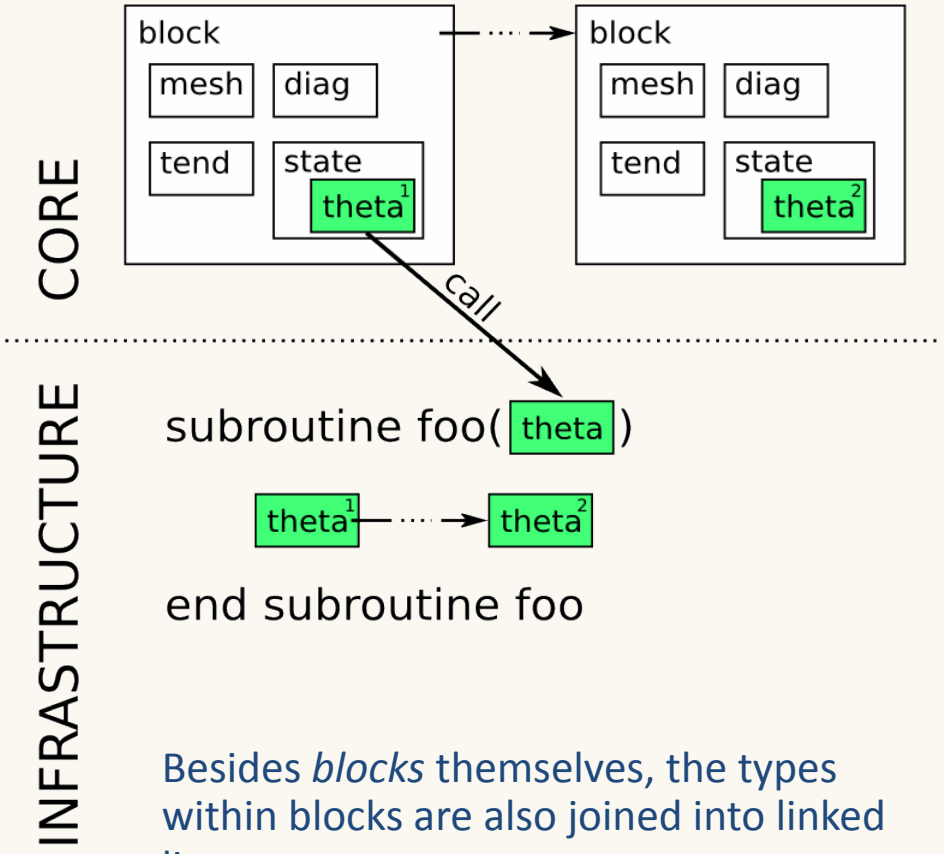
Applying a broader interpretation of *core*, we envision building pre- and post-processing, or MPAS analysis software, within the MPAS framework as well

- Actually, a separate real-data initialization core for the non-hydrostatic model is currently under construction!

MPAS MODEL CORE (2)



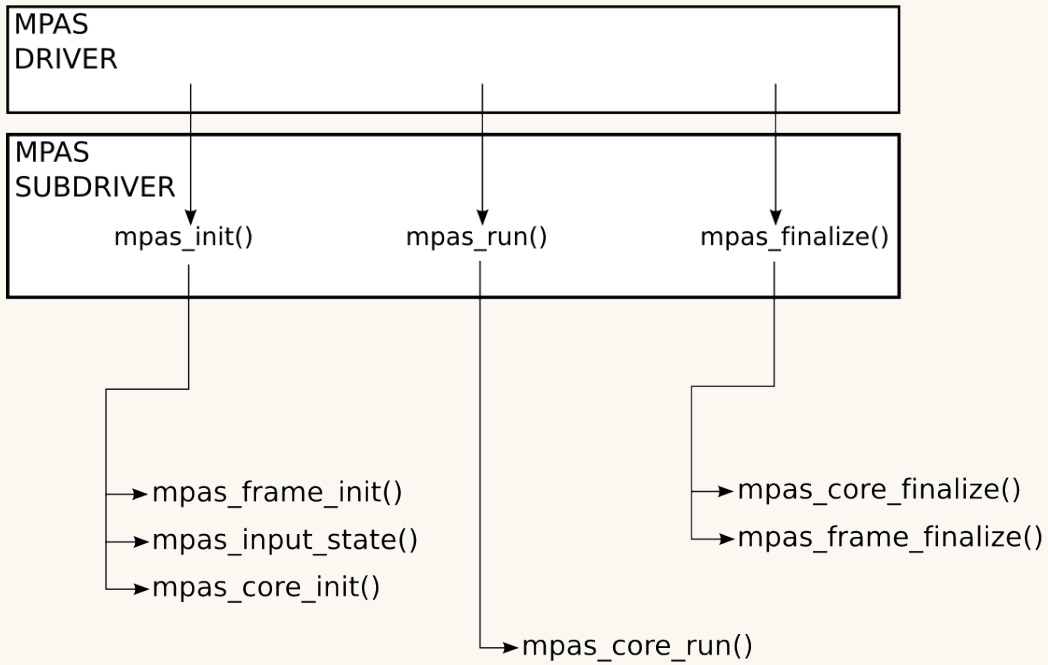
The developer of an MPAS *core* must take care to properly support multiple blocks per process



Besides *blocks* themselves, the types within blocks are also joined into linked lists

Infrastructure routines can be called with the, e.g., the *field* from the head of the list, and all blocks for that field can be operated upon

THE DRIVER LAYERS



The SUBDRIVER interacts with both the INFRASTRUCTURE and CORE, and takes care of "boilerplate" code that would otherwise be duplicated in all MPAS cores.

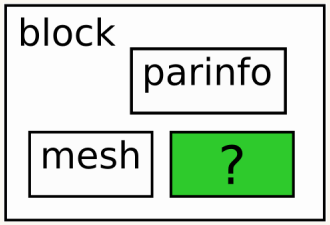
The choice to split the superstructure into two layers was motivated by

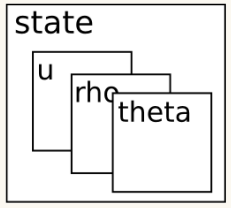
- The convenience of placing core-agnostic, driver-like code into one common place
- The potential need to replace the highest-level driver code with, e.g., and interface layer when driving MPAS from within other earth system models

THE MPAS REGISTRY

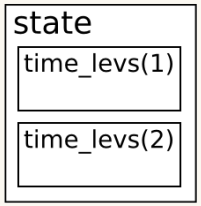
The need to support different cores in the MPAS framework suggests that the developer of a core would need to write "copy-and-paste" code to handle aspects such as the definition, allocation/deallocation, and I/O for each of the fields needed by the core; this would be especially true in MPAS given that we permit

- grouping of fields within higher-level data structures
- fields to have differing numbers of time levels (e.g., two levels for prognostic fields, a single level for diagnostic fields)
- aggregation of fields into a "super-field" (e.g., combining qv, qc, and qr into a single moist_scalars field)
- I/O streams to be specified on a field-by-field basis

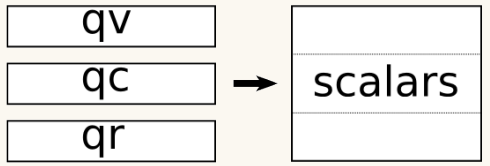





```
state % u
state % rho
...
```




```
state % time levs(:) % state % theta
```

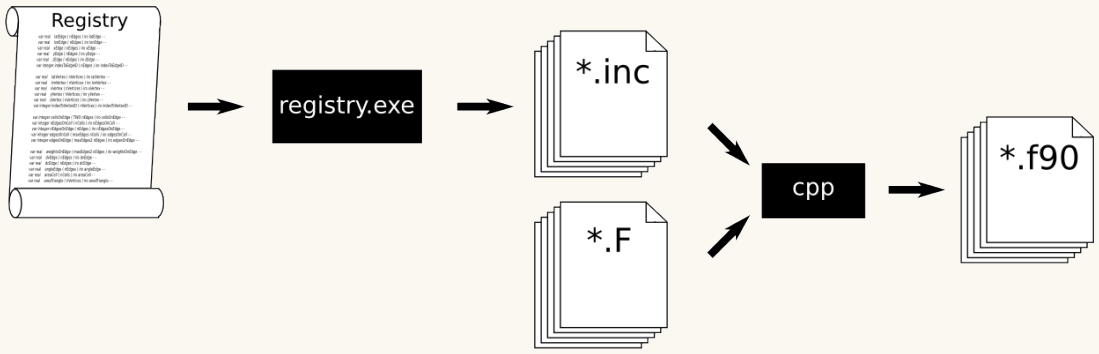



```
qv(:,:)  → scalars(idx_qv,:,:)
```

THE MPAS REGISTRY

As an answer to the challenges just posed, we have employed a “Registry” as part of our prototype MPAS framework

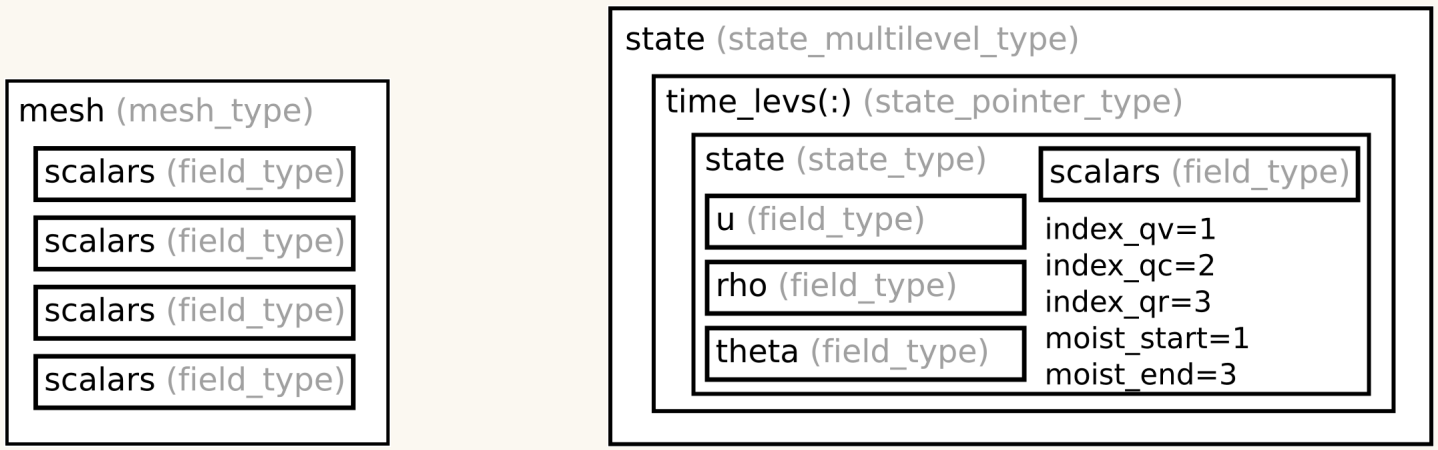
- An idea borrowed from WRF (Michalakes (2004))
- The Registry is a “data dictionary”: each field has an entry providing meta-data and other attributes (type, dims, I/O streams, etc.)
- Each MPAS core is paired with its own registry file
- At compile time, a small C program is first compiled; the program runs, parses registry file, and generates Fortran code
  - Among other things, creates code to allocate, read, and write fields



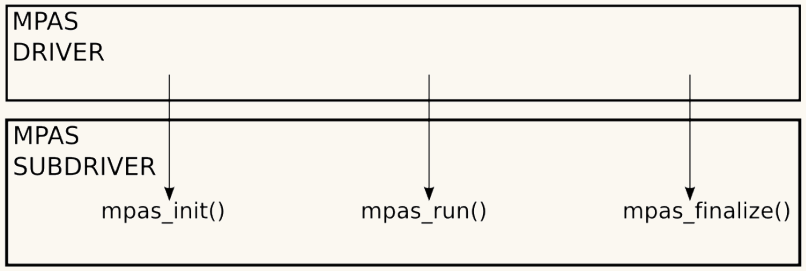
For dynamics-only non-hydrostatic atmosphere model, Registry generates ~8900 lines of code vs 6600 lines hand-written for dynamics

EX: THE NON-HYDROSTATIC REGISTRY

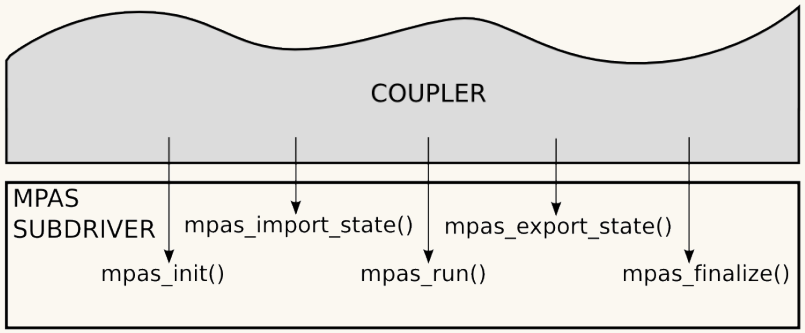
```
# Horizontal grid structure:
var persistent real    latCell ( nCells ) 0 iro latCell mesh - -
var persistent real    lonCell ( nCells ) 0 iro lonCell mesh - -
var persistent real    latEdge ( nEdges ) 0 iro latEdge mesh - -
var persistent real    lonEdge ( nEdges ) 0 iro lonEdge mesh - -

# Prognostic variables:
var persistent real    u ( nVertLevels nEdges Time ) 2 iro u state - -
var persistent real    rho ( nVertLevels nCells Time ) 2 iro rho state - -
var persistent real    theta ( nVertLevels nCells Time ) 2 iro theta state - -
var persistent real    qv ( nVertLevels nCells Time ) 2 iro qv state scalars moist
var persistent real    qc ( nVertLevels nCells Time ) 2 iro qc state scalars moist
var persistent real    qr ( nVertLevels nCells Time ) 2 iro qr state scalars moist
```



COUPLING VIA THE DRIVER LAYERS



The current DRIVER layer in MPAS simply calls *init*, *run*, and *finalize* routines implemented by the SUBDRIVER layer



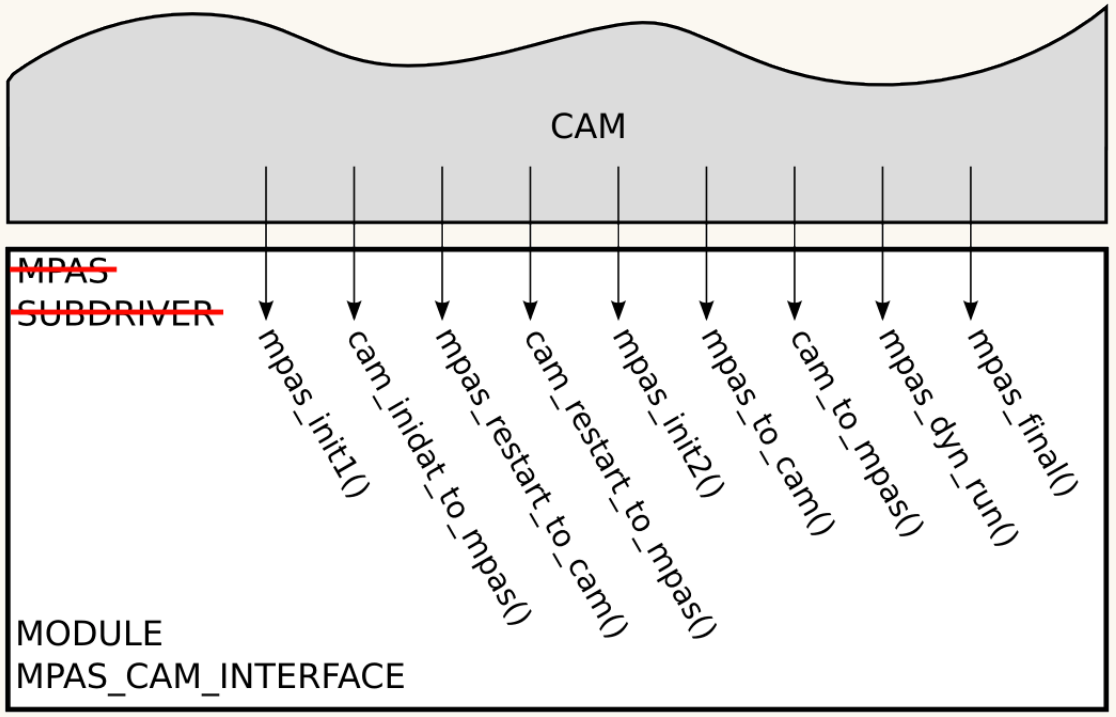
The driver can be replaced by a coupler that calls (possibly additional) routines implemented in the SUBDRIVER

EX: MPAS HYDROSTATIC CORE IN CAM

Work with LLNL colleagues to use MPAS hydrostatic atmosphere core as a dynamical core in CAM

MPAS development essentially involved re-implementing the SUBDRIVER module with CAM-specific interface routines

- *Some generalization to MPAS routines (e.g., optional MPI communicator argument to parallelism infrastructure init)*



Import state	Export state	Control
cam_inidat_to_mpas cam_restart_to_mpas cam_to_mpas	mpas_restart_to_cam mpas_to_cam	mpas_init1 mpas_init2 mpas_dyn_run mpas_final

COUPLING VIA I/O INFRASTRUCTURE

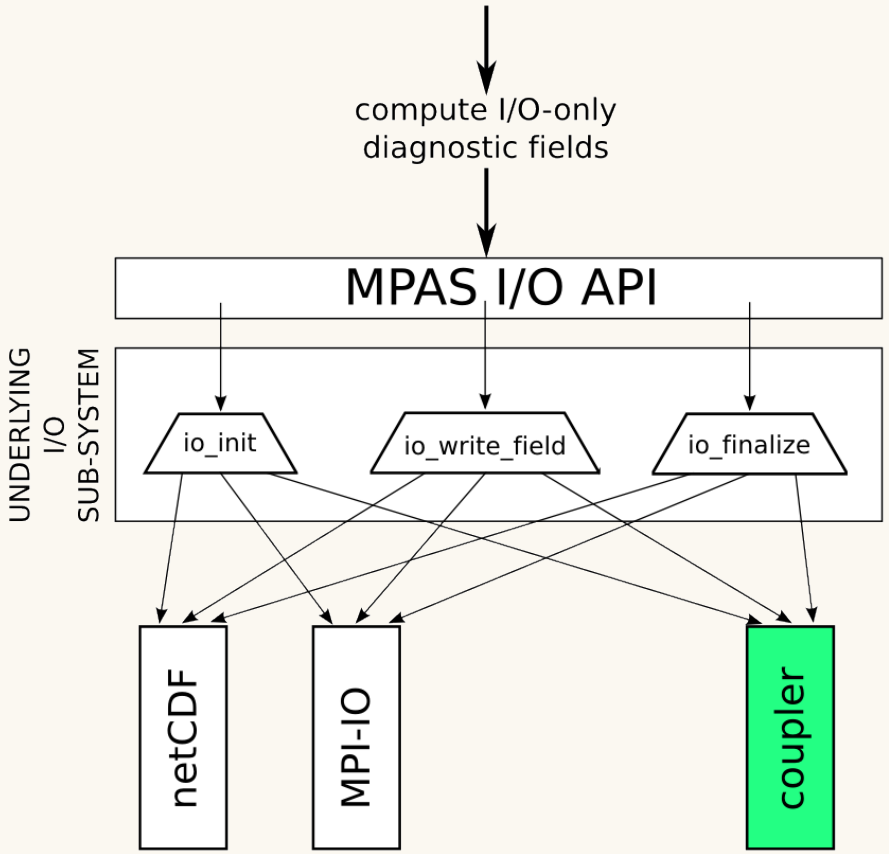
The envisioned design of the I/O layer would provide a single interface for

- Serial and parallel I/O
- Different file formats

A coupler that uses get/put calls might then be made to appear as just another I/O format available through the high-level I/O API

- Not a novel idea – see, e.g., WRF's I/O API (Michalakes et al. (2004))
- OASIS/PRISM and MCT through PIO?

If MPAS I/O were easily controlled (e.g., by a configuration file), coupling any model built within the MPAS framework would become "trivial"



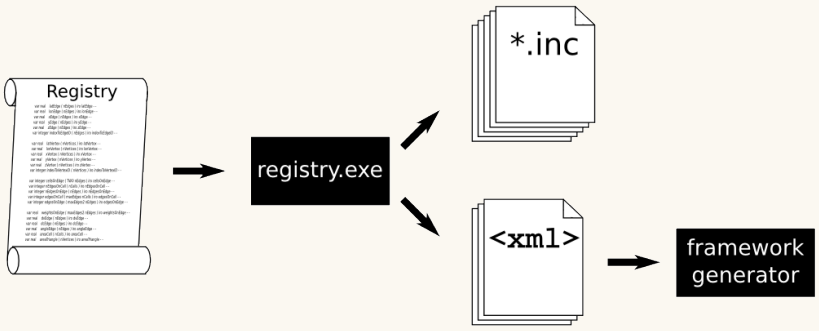
ROLE OF THE REGISTRY IN COUPLING

The registry can generate more than just Fortran code – anything we'd like it to generate based on registry entries, in fact!

- Including information for metadata-driven couplers or framework generators

The syntax of the MPAS registry files is easily changed or updated

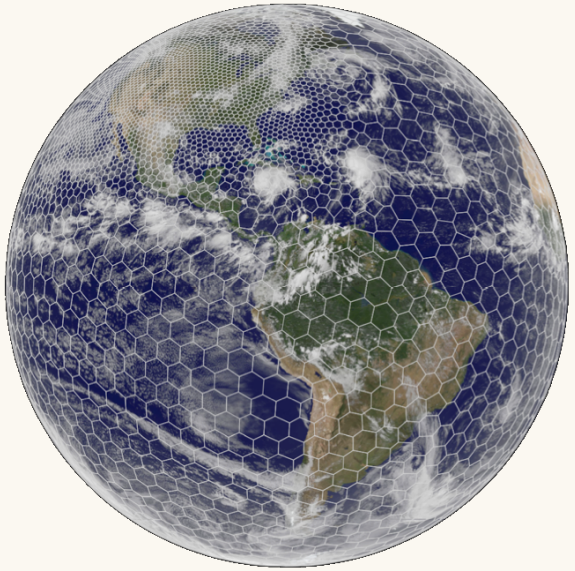
- Could be extended to permit additional attributes and metadata
  - ESMF Attributes?
- Could be changed to another format, e.g., XML

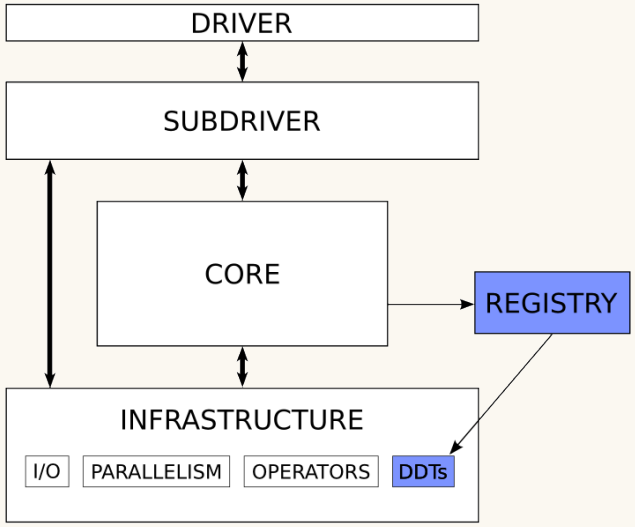



```
field {
        name:u
        dimensions:nVertLevels,nCells
        units:"m s-1"
        description:"normal velocity on cell faces"
        coupled-write:true
        coupled-read:needed
}
```

CONCLUSIONS

- MPAS is comprised of several earth system component models sharing a common software framework; if done correctly, the work of coupling one MPAS model can be re-used by any MPAS model
- Any coupling framework used in MPAS must support fully-unstructured grids (in the horizontal)
- The *Registry* mechanism in MPAS could be leveraged to maintain additional meta-data, and to generate information for other framework generation code
- Multi-language APIs are desirable
  - Note how naturally the MPAS architecture maps onto an object-oriented design
- The flexibility of MPAS meshes, plus the existence of atmosphere and ocean cores in the same framework, might make MPAS useful for exercising coupling frameworks





REFERENCES

Du, Q., V. Faber and M. Gunzburger, 1999, Centroidal Voronoi tessellations: Applications and algorithms, SIAM Review, 41, 637-676.

Du, Q., M. Gunzburger and L. Ju, 2003, Constrained centroidal Voronoi tessellations for surfaces, SIAM Journal on Scientific Computing, 24, 1488-1506.

Ju, L., T. Ringler and M. Gunzburber, 2010, Voronoi Diagrams and Application in Climate and Global Modeling, Numerical Techniques for Global Atmospheric Models, Lecture Notes in Computational Science, draft.

Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, 2004: "The WeatherReseach and Forecast Model: Software Architecture and Performance," to appear in proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology, 25-29 October 2004, Reading, U.K. Ed. George Mozdzynski.

Ringler, T., L. Ju and M. Gunzburger, 2008, A multiresolution method for climate system modeling: application of spherical centroidal Voronoi tessellations, Ocean Dynamics, 58 (5-6), 475-498.