```
netcdf grid {
dimensions:
        nCells = 40962 ;  this is the number of mass cells
        nEdges = 122880 ;  this is the number of velocity points
        nVertices = 81920 ;  this is the number of vorticity cells
        maxEdges = 10 ;  maximum number of edges that any one cell can have
        maxEdges2 = 20 ;  2*maxEdges
        TWO = 2 ;
        THREE = 3 ;
        nVertLevels = 1 ;  this is the number of vertical levels (set here to one for testing)
        nTracers = 5 ;  this is the number of tracers (again, set to 5 for testing)
        Time = UNLIMITED ; // (1 currently)
variables:
        double latCell(nCells) ;  the center of each cell in latitude
        double lonCell(nCells) ; the center of each cell in longitude
        double xCell(nCells) ;  the x position of the cell center on the unit sphere
        double yCell(nCells) ;  the y position of the cell center on the unit sphere
        double zCell(nCells) ;  the z position of the cell center on the unit sphere
        int indexToCellID(nCells) ; the unique integer tag for each cell (ranges 1 to nCells)
        double latEdge(nEdges) ;  the position of the velocity points in latitude
        double lonEdge(nEdges) ; the position of the velocity points in longitude
        double xEdge(nEdges) ; the x position of the cell edge center on the unit sphere
        double yEdge(nEdges) ; the y position of the cell edge center on the unit sphere
        double zEdge(nEdges) ; the z position of the cell edge center on the unit sphere
        int indexToEdgeID(nEdges) ; the unique integer tag for each edge
        double latVertex(nVertices) ; the position of the vorticity points in latitude
        double lonVertex(nVertices) ; the position of the vorticity points in longitude
        double xVertex(nVertices) ; the x position of the cell vertices on the unit sphere
        double yVertex(nVertices) ; the y position of the cell vertices on the unit sphere
        double zVertex(nVertices) ; the z position of the cell vertices on the unit sphere
        int indexToVertexID(nVertices) ; the unique integer tag for each vertex
        int cellsOnEdge(nEdges, TWO) ;  the cell indices that share iEdge
        int nEdgesOnCell(nCells) ; the number of edges for iCell
        int nEdgesOnEdge(nEdges) ; number of edges in cells sharing iEdge
        int edgesOnCell(nCells, maxEdges) ; the edge indices associated with iCell
        int edgesOnEdge(nEdges, maxEdges2) ; indices for edges in cells sharing iEdge
        double weightsOnEdge(nEdges, maxEdges2) ;  weights for velocity reconstruction
        double dvEdge(nEdges) ;  the distance between the vertices at the ends of iEdge
        double dv1Edge(nEdges) ;  (not used at present, filled with zeros)
        double dv2Edge(nEdges) ; (not used at present, filled with zeros)
        double dcEdge(nEdges) ; the distance between the cells that share iEdge
        double angleEdge(nEdges) ;  the angle the edges makes with the east direction
        double areaCell(nCells) ;  area of each cell (where mass lives)
        double areaTriangle(nVertices) ;  area of each triangle (where vorticity lives)
        int cellsOnCell(nCells, maxEdges) ; the cell indices of iCell that share an edge
        int verticesOnCell(nCells, maxEdges) ; vertex indices associated with iCell
         int verticesOnEdge(nEdges, TWO) ;  vertex indices associated with iEdge
         int edgesOnVertex(nVertices, THREE) ;  edge indices associated with iVertex
         int cellsOnVertex(nVertices, THREE) ;  cells indices associated with iVertex
         double kiteAreasOnVertex(nVertices, THREE) ;  (labeled on Figure 1)
         double fEdge(nEdges) ; Coriolis parameter evaluated at edge points
         double fVertex(nVertices) ; Coriolis parameter evaluated at vertex points
```

location of edge points (velocity points)

centers of dual-mesh cells (vorticity points)

centers of primal-mesh cells (mass points)

We generally refer to mass points as "cells" and refer to vorticity points as "vertices" and refer to velocity points as "edges".

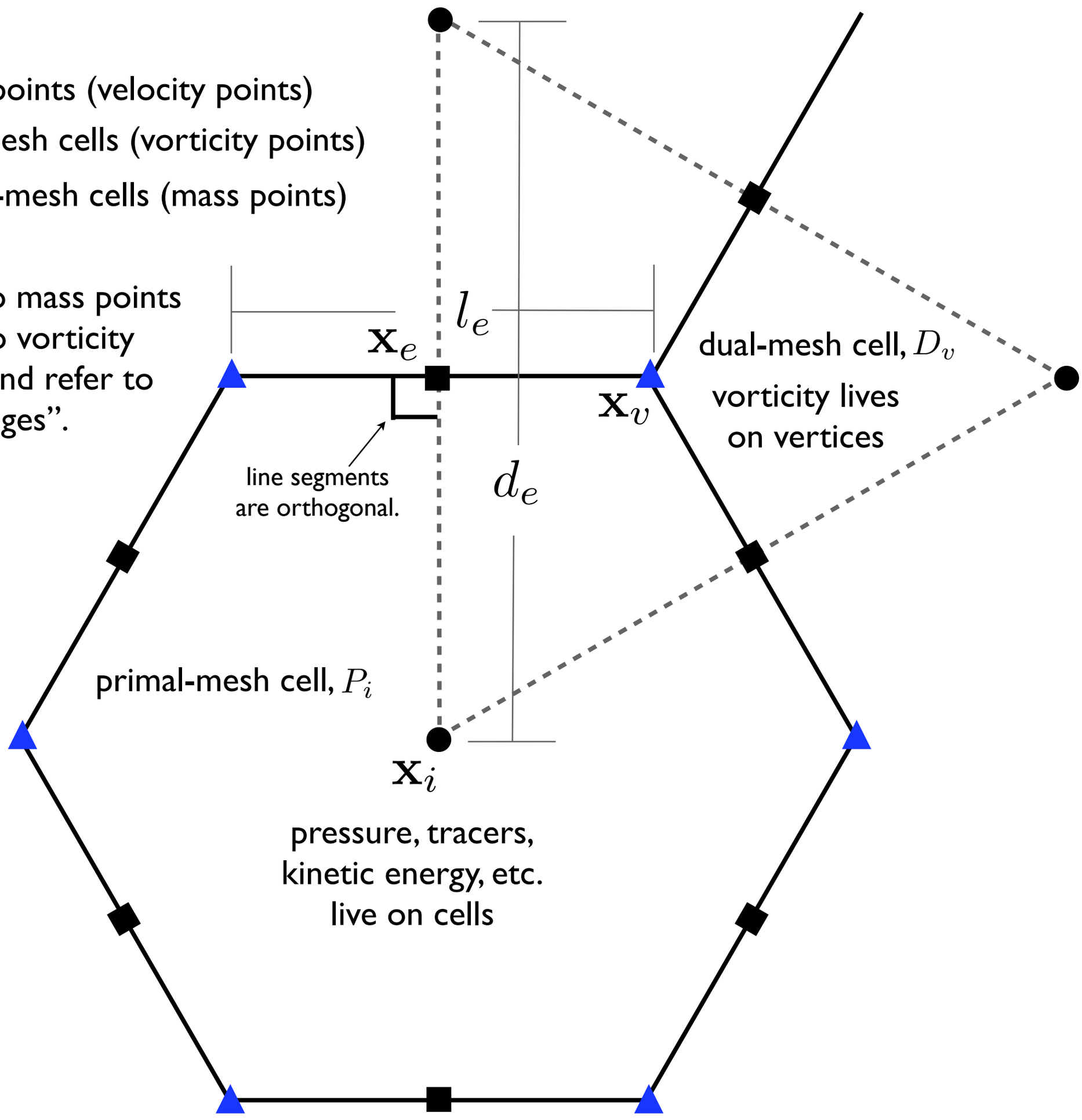
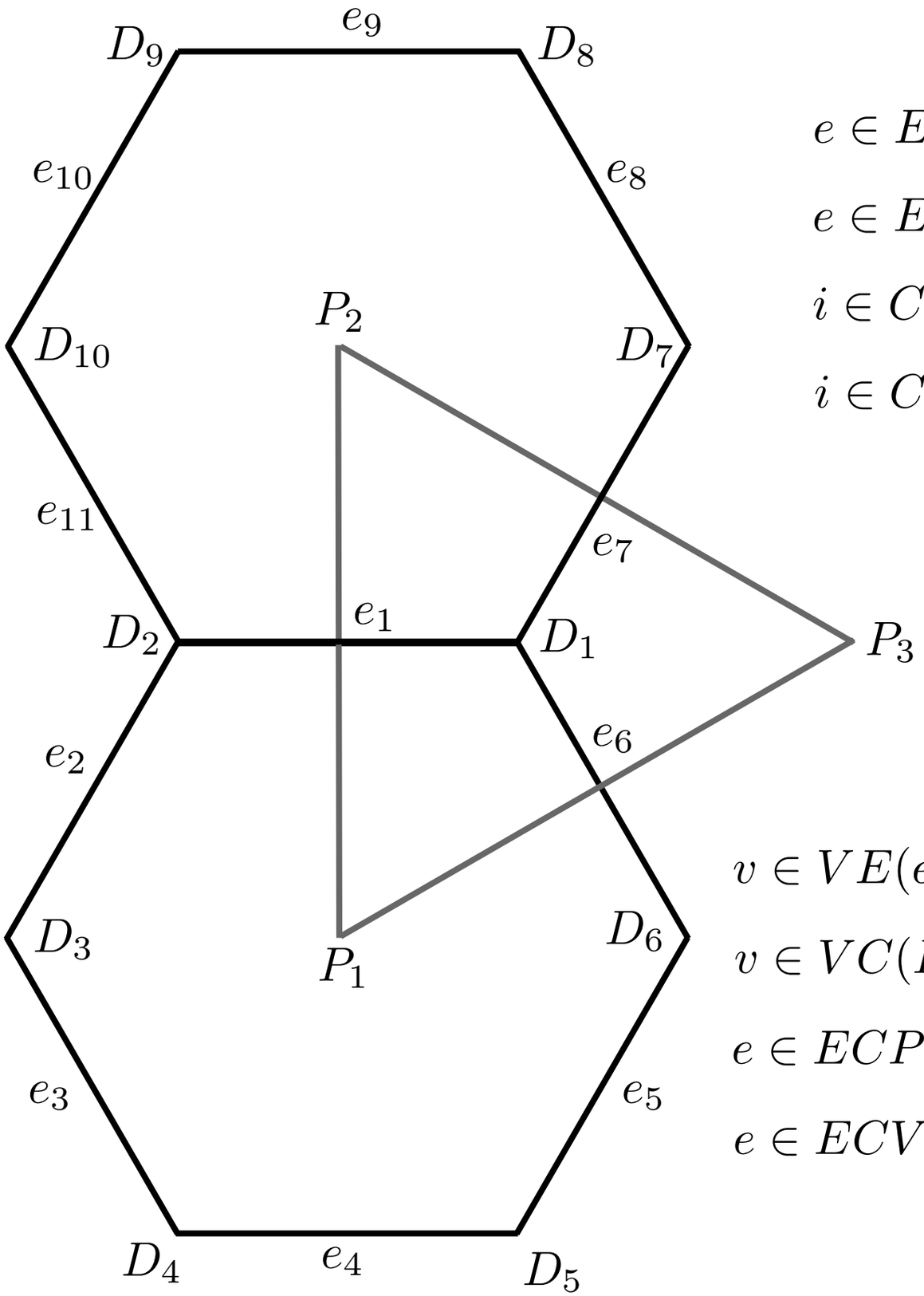


Figure 1



$e \in EC(P_1) = [e_1, e_2, e_3, e_4, e_5, e_6]$ edgesOnCell

$e \in EV(D_1) = [e_1, e_6, e_7]$ edgesOnVertex

$i \in CE(e_1) = [P_1, P_2]$ cellsOnEdge

$i \in CV(D_1) = [P_1, P_2, P_3]$ cellsOnVertex

$v \in VE(e_1) = [D_1, D_2]$ VerticesOnEdge

$v \in VC(P_1) = [D_1, D_2, D_3, D_4, D_4, D_5, D_6]$ VerticesOnCell

$e \in ECP(e_1) = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}]$

EdgesOnEdge

$e \in ECV(P_1, D_1) = [e_1, e_6]$ (not in grid.nc)

Figure 2

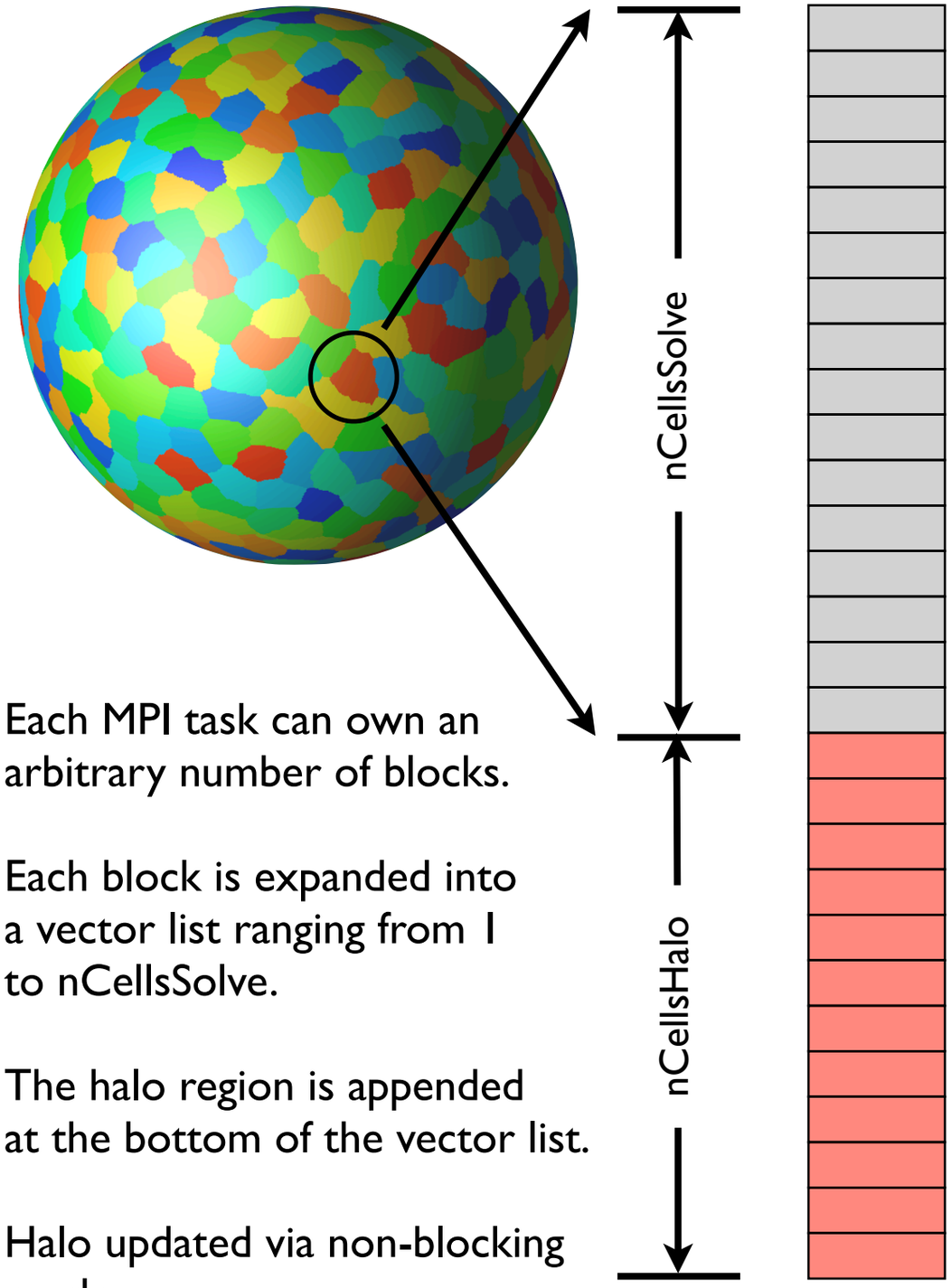
Decomposition for implementation on MPPs.

Horizontal decomposition is done with METIS.

At present, each processor gets one block. In the future, each processor might own multiple blocks to improve cache reuse.

Figure 3

Data Layout: Unstructured in horizontal, structured in vertical.



Each MPI task can own an arbitrary number of blocks.

Each block is expanded into a vector list ranging from 1 to nCellsSolve.

The halo region is appended at the bottom of the vector list.

Halo updated via non-blocking sends.

nCellsTotal = nCellsSolve + nCellsHalo

The ordering of the cells is determined by Reverse Cuthill-McKee (RCM) to maximize cache reuse.

The data access pattern in repeated in the vertical leading to arrays dimensioned as

mass(nVertLevels, nCellsTotal, nBlocks)

Example: sum mass at neighbors cells, each proc executes

```
do iBlock = 1, nBlocks
 do i = 1 , nCellsSolve(iBlock)
  do j=1,nCellsOnCell(i,iBlock)
       coc = cellsOnCell(i,j,iBlock)
       do k=1,nVertLevels
         r(k,i) = r(k,i) + mass(k,coc,iBlock)
       enddo
  enddo
 enddo
endo
```

The vertical loop is always the inner most loop to vectorize and to limit indirect addressing.

The layout is repeated for variables that live at vertices and at edges.

What data is local?

Assume that P1 is a real cell and that P2 is a halo cell.

All data that lives on P2 is present.

From the perspective of a loop over iCell:
- Vertex data D7 through D10 is valid.
- Edge data e7 through e11 is valid.

From the perspective of a loop over iVertex
- If one of the cellsOnVertex is a real cell (e.g. D1)
  - then all data in cellsOnVertex, edgesOnVertex, and verticesOnVertex is valid.
- If none of the cellsOnVertex is a real cell (e.g. D7),
  - then only data associated with the distance-one neighbors (e.g. P2) is valid.

From the perspective of a loop over iEdge
- If one of the cellsOnEdge is a real cell (e.g. e1)
  - then all data in cellsOnEdge, verticesOnEdge, and edgesOnEdge is valid.
- If none of the cellsOnEdge is a real cell (e.g. e8),
  - then only data associated with the distance-one neighbors (e.g. P2) is valid.