Representing Simple Geometry Types in NetCDF-CF
(IN23A-1757)

David L. Blodgett (USGS), Ben Koziol (NOAA / Univ. of Colorado), Timothy Whiteaker (CRWR / Univ. of Texas-Austin), Robert Simons (NOAA/NMFS/SWFSC)

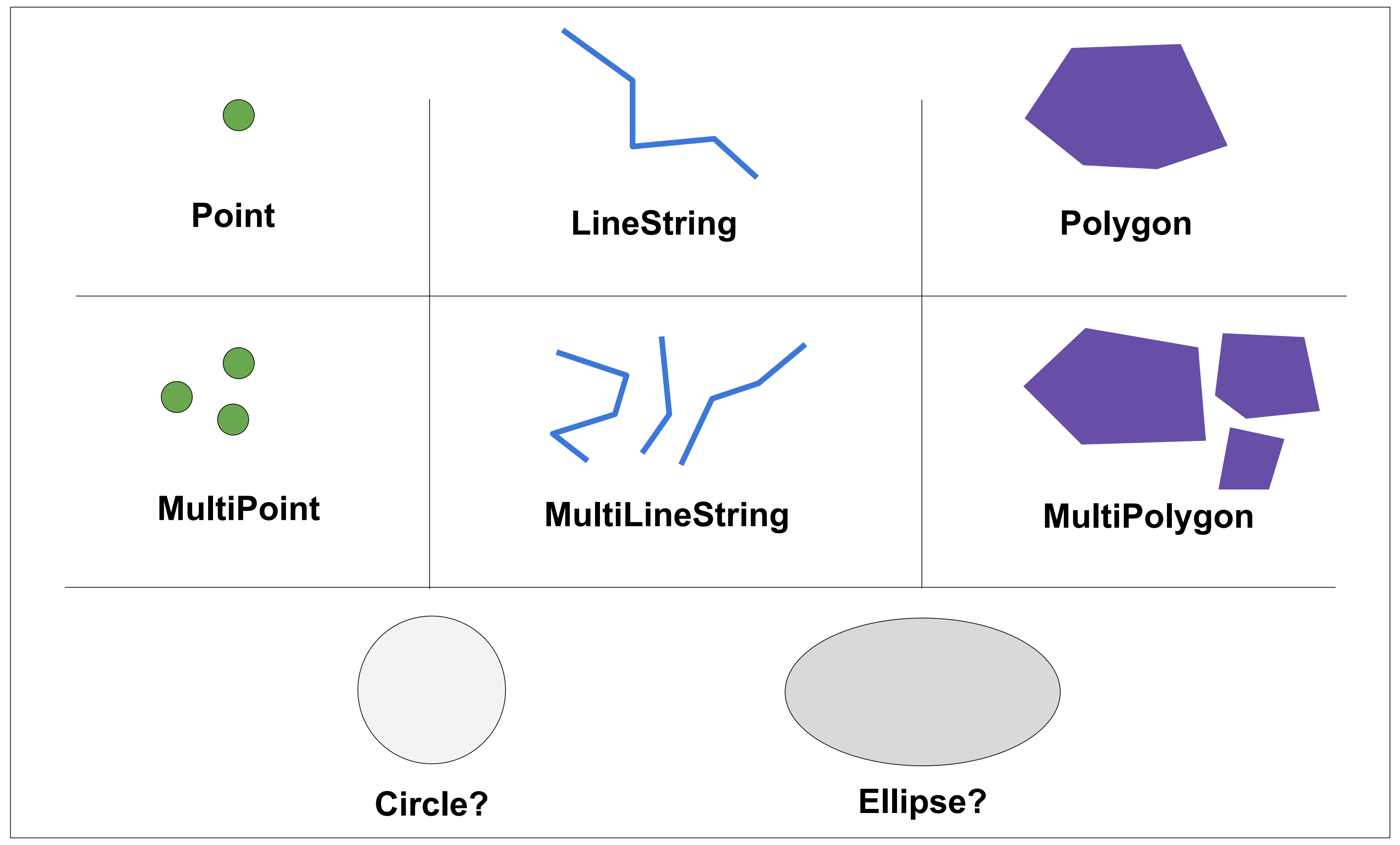
See this poster at: https://goo.gl/0NI4Sd Contact: dblodgett@usgs.gov

NetCDF - Climate & Forecast (CF) Convention Primer

- **NetCDF** → Network Common Data Form → “NetCDF is a set of [primarily open source] software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.” (http://www.unidata.ucar.edu/software/netcdf/)
- **CF** → Climate and Forecast Metadata Convention → “The conventions for CF metadata are designed to promote the processing and sharing of files created with the NetCDF API. The CF conventions are increasingly gaining acceptance and have been adopted by a number of projects and groups as a primary standard. The conventions define metadata that provide a definitive description of what the data in each variable represents, and the spatial and temporal properties of the data. This enables users of data from different sources to decide which quantities are comparable, and facilitates building applications with powerful extraction, regridding, and display capabilities.” (http://cfconventions.org/)

What is a “simple geometry”?

- By “simple geometry” we mean points, lines, polygons, and their multipart equivalents. Lines and polygon edges are composed of straight line segments. In other words, we do not support curves, though curves could be approximated by linear segments.
- Multiparts are used when a single feature consists of several parts, such as the State of Hawaii (multipolygon) or a sensor array (multipoint) generating a composite measurement for a test site. On the other hand, for a river network where each river segment has its own stream discharge data, the river segments would likely be represented by single part line geometries rather than a lone multiline for the entire river network.
- These geometry types are modeled after the Well-Known Text (WKT) geometry primitives and multipart equivalents defined in the ISO/IEC 13249-3:2016 standard. These types are compatible with commonly used geospatial formats and software including GeoJSON, shapefile, ArcGIS, QGIS, and Shapely.
- We would consider supporting curves or parametric shapes such as ellipses if there is enough demand for this functionality. Let us know what you think!



Storing Geometries in NetCDF - State of the Art 2016

Unstructured Grid Convention (UGRID)

UGRID provides a comprehensive method for encoding unstructured grids using regular shapes (triangles), lines, and flexible meshes (arbitrary polygons) with two and three dimensional coordinates. UGRID was designed for NetCDF and is compatible with CF. UGRID has no built-in methods for handling points and multi-geometries. UGRID is the primary inspiration for the simple geometries approach.

- http://ugrid-conventions.github.io/ugrid-conventions/
- https://github.com/pyugrid/pyugrid

CF Convention Discrete Sampling Geometries (DSG)

DSG handles data from one or more TimeSeries, Trajectory, Profile, TrajectoryProfile or TimeSeriesProfile. DSG is sufficient for representing time series at a point. DSG has no system to define a geometry (point, polyline, polygon, etc.). DSG is part of the current CF specification and efforts will be made to be compatible with DSG and reuse controlled vocabulary.

- http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html#discrete-sampling-geometries

Open Geospatial Consortium (OGC) / Esri

OGC and Esri standards provide the default storage mechanisms for GIS data. These data are linked to NetCDF-based data using matching unique identifiers in the NetCDF file and GIS dataset when there is no NetCDF convention for storing the geometry from the GIS.

- http://www.opengeospatial.org/docs/is

Justification & Motivation

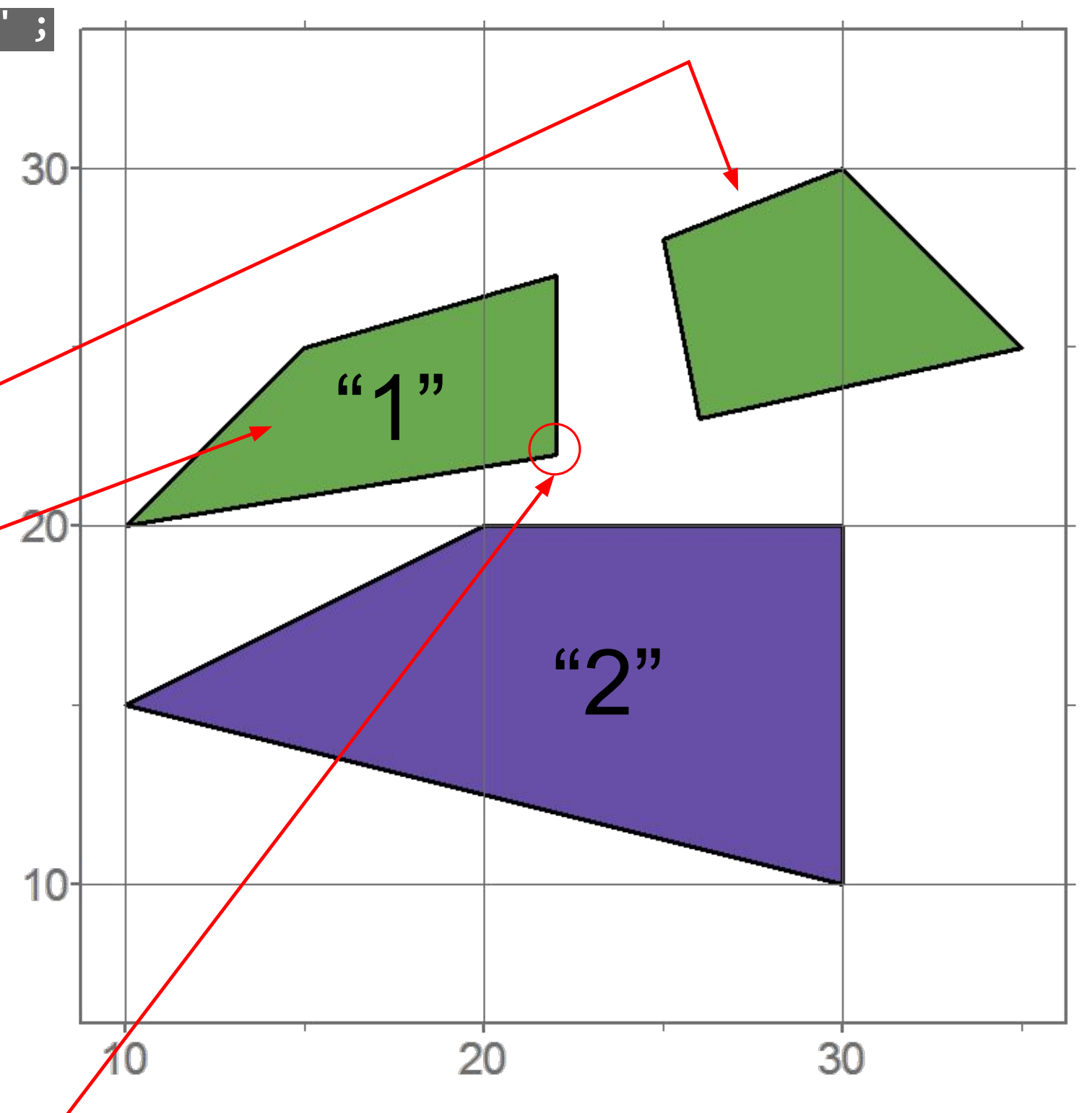
- CF currently has no accepted mechanism for representing simple geometry types such as lines and polygons.
- Lack of support for simple geometries within CF has unintentionally excluded a broad set of geoscientific data types from NetCDF-CF data encodings.
- Working with large scientific GIS datasets is difficult with current NetCDF metadata conventions, often yielding an unwieldy hybrid of NetCDF data and other software like Esri ArcGIS and PostGIS.
- For example, the US National Weather Service’s National Water Model (NWM) outputs stream discharge forecasts for about 2.7 million stream segments throughout the continental US in NetCDF-CF format. Because there is no CF convention for representing line geometries such as river segments, NWM output is linked to river geometry using the upstream endpoint of each river segment, i.e., a point instead of a line.

Proposed Encoding Method for Climate & Forecast Convention Simple Geometries

Common Data Language (CDL):

Highlighted text shows newly proposed CF attributes.

```
netcdf sample_poly {
dimensions:
   char = 1 ;
   instance = 2 ;
   coordinate_index = 16 ;
   coordinates = 15 ;
variables:
   char instance_name(instance, char) ;
      instance_name:standard_name = "instance_id" ;
   int coordinate_index(coordinate_index) ;
      coordinate_index:long_name = "ragged index for coordinates and geometry break values" ;
      coordinate_index:geom_coordinates = "x y" ;
      coordinate_index:geom_dimension = "instance" ;
      coordinate_index:start_index = 1 ;
      coordinate_index:stop_encoding = “cra” ;
      coordinate_index:outer_ring_order = "clockwise" ;
      coordinate_index:closure_convention = "last_node_equals_first" ;
      coordinate_index:geom_type = "multipolygon" ;
      coordinate_index:multipart_break_value = -1 ;
   int coordinate_index_stop(instance) ;
      coordinate_index_stop:long_name = "index for last coordinate in each instance geometry" ;
      coordinate_index_stop:contiguous_ragged_dimension = "coordinate_index" ;
   double x(coordinates) ;
      x:units = "degrees_east" ;
      x:standard_name = "longitude" ; // or projection_x_coordinate
      X:cf_role = "geometry_x_node" ;
   double y(coordinates) ;
      y:units = "degrees_north" ;
      y:standard_name = “latitude” ; // or projection_y_coordinate
      y:cf_role = "geometry_y_node"

// global attributes:
      :Conventions = "CF-1.8" ;
data:
 instance_name =
  "1",
  "2" ;

 coordinate_index = 1, 2, 3, 4, 5, -1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 ;

 coordinate_index_stop = 11, 16 ;

 x = 35, 26, 25, 30, 35, 22, 10, 15, 22, 22, 30, 10, 20, 30, 30 ;

 y = 25, 23, 28, 30, 25, 22, 20, 25, 27, 22, 10, 15, 20, 20, 10 ;
```



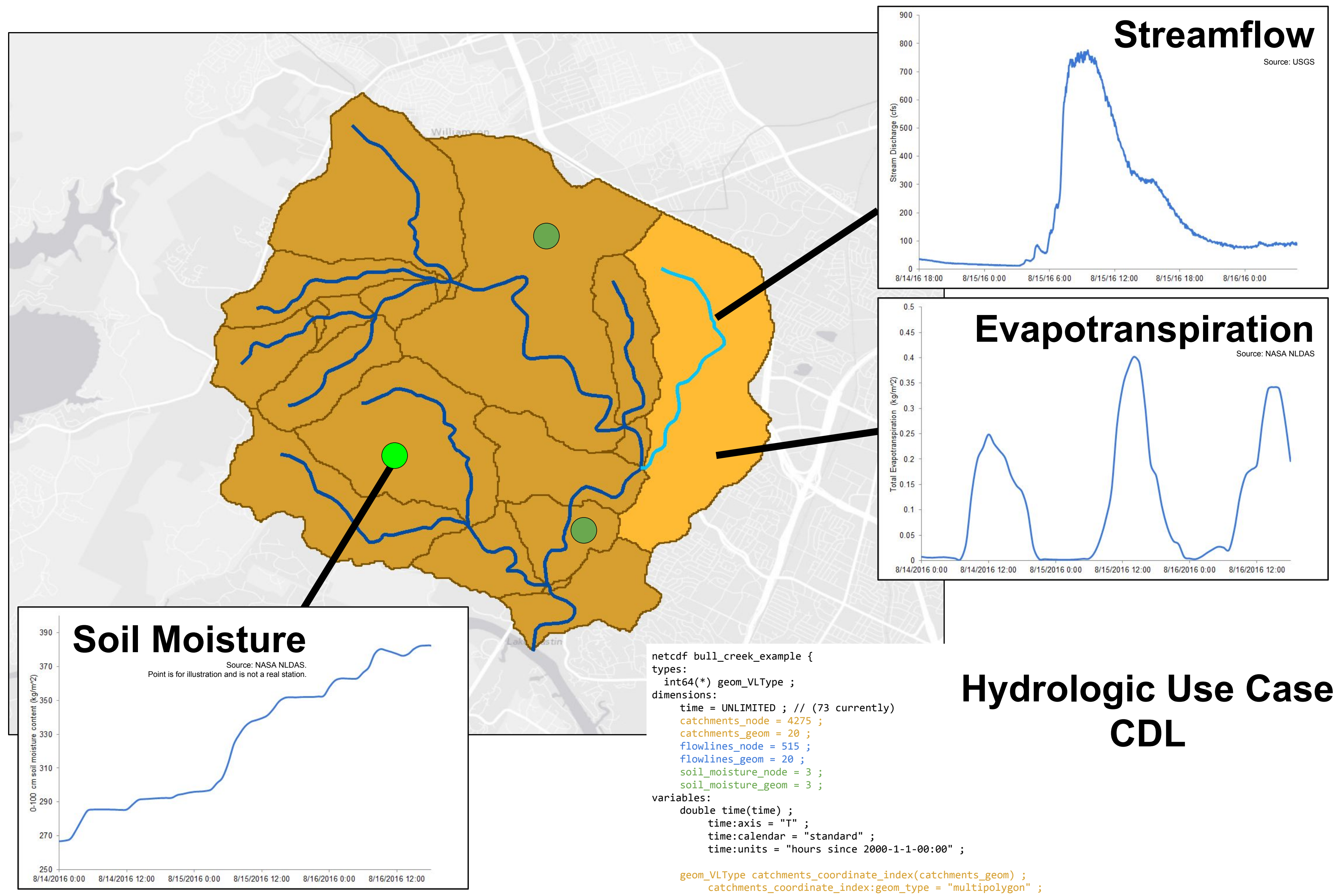
Well Known Text (WKT):

```
GEOMETRYCOLLECTION(MULTIPOLYGON (((35 25, 26 23, 25 28, 30 30, 35 25)), ((22 22, 22 27, 10 20, 15 25, 22 22))),
                   POLYGON ((30 10, 10 15, 20 20, 30 20, 30 10)))", id = c("1","2"))
}
```

Contiguous Ragged & Variable Length Arrays

- Line, polygon, and multipart geometry coordinate data vary in length depending on per-feature coordinate counts. This presents a difficulty when storing them in the multidimensional arrays used by NetCDF since memory for the full array dimensionality must be pre-allocated when creating variables. Each geometry consumes as much memory as the largest node count in the collection.
- This issue is addressed in NetCDF-CF using contiguous ragged arrays and is handled with a new Variable Length data type in the HDF5-based NetCDF4 data model.
- **Contiguous** → Contiguous ragged arrays are used to store tabular data where data “columns” have varying length. This approach requires one large data vector and a second shorter vector indicating how the data vector should be split.
  - The encoding method example above demonstrates contiguous ragged arrays for multipolygon features. This approach is very similar to current NetCDF-CF.
  - This method accounts for “break values” between multipart geometries and/or holes using attributes to indicate the values of these breaks.
  - The “contiguous_ragged_dimension” indicates which shared dimension the stop variable indexes.
- **Variable Length** → Variable length arrays, which are supported by HDF5 and the NetCDF4 API, offer a solution where variable length tabular data such as geometry coordinates can be stored without unnecessary memory allocations. The hydrologic use case in the upper right demonstrates NetCDF4 Variable Length arrays encoding geometry data.

Hydrologic Use Case for Simple Geometries



Hydrologic Use Case CDL

```
netcdf bull_creek_example {
types:
  int64(*) geom_VLType ;
dimensions:
     time = UNLIMITED ; // (73 currently)
     catchments_node = 4275 ;
     catchments_geom = 20 ;
     flowlines_node = 515 ;
     flowlines_geom = 20 ;
     soil_moisture_node = 3 ;
     soil_moisture_geom = 3 ;
variables:
     double time(time) ;
          time:axis = "T" ;
          time:calendar = "standard" ;
          time:units = "hours since 2000-1-1-00:00" ;

     geom_VLType catchments_coordinate_index(catchments_geom) ;
          catchments_coordinate_index:geom_type = "multipolygon" ;
          catchments_coordinate_index:geom_dimension = "catchments_geom" ;
          catchments_coordinate_index:geom_coordinates = "catchments_x catchments_y" ;
          catchments_coordinate_index:stop_encoding = "vlen" ;
          catchments_coordinate_index:outer_ring_order = "anticlockwise" ;
          catchments_coordinate_index:closure_convention = "last_node_equals_first" ;
     double catchments_x(catchments_node) ;
          Catchments_x:cf_role = "geometry_x_node" ;
     double catchments_y(catchments_node) ;
          catchments_y:cf_role = "geometry_y_node" ;
     double catchments_evapotranspiration(time, catchments_geom) ;
         catchments_evapotranspiration:long_name = "total evapotranspiration" ;
         catchments_evapotranspiration:units = "kg/m^2" ;

     geom_VLType flowlines_coordinate_index(flowlines_geom) ;
          flowlines_coordinate_index:geom_type = "linestring" ;
          flowlines_coordinate_index:geom_dimension = "flowlines_geom" ;
          flowlines_coordinate_index:geom_coordinates = "flowlines_x flowlines_y flowlines_z" ;
          flowlines_coordinate_index:stop_encoding = "vlen" ;
     double flowlines_x(flowlines_node) ;
          flowlines_x:cf_role = "geometry_x_node" ;
     double flowlines_y(flowlines_node) ;
          flowlines_y:cf_role = "geometry_y_node" ;
     double flowlines_z(flowlines_node) ;
          flowlines_z:cf_role = "geometry_z_node" ;
     double flowlines_streamflow(time, flowlines_geom) ;
         flowlines_streamflow:standard_name = "water_volume_transport_in_river_channel" ;
         flowlines_streamflow:units = "ft^3/sec" ;

     geom_VLType soil_moisture_coordinate_index(soil_moisture_geom) ;
          soil_moisture_coordinate_index:geom_type = "point" ;
          soil_moisture_coordinate_index:geom_dimension = "soil_moisture_geom" ;
          soil_moisture_coordinate_index:geom_coordinates = "soil_moisture_x soil_moisture_y soil_moisture_z" ;
          soil_moisture_coordinate_index:stop_encoding = "vlen" ;
     double soil_moisture_x(soil_moisture_node) ;
          soil_moisture_x:cf_role = "geometry_x_node" ;
     double soil_moisture_y(soil_moisture_node) ;
          soil_moisture_y:cf_role = "geometry_y_node" ;
     double soil_moisture_z(soil_moisture_node) ;
          soil_moisture_z:cf_role = "geometry_z_node" ;
     double soil_moisture(time, soil_moisture_geom) ;
         soil_moisture:standard_name = "moisture_content_of_soil_layer" ;
         soil_moisture:units = "kg/m^2" ;

// global attributes:
          :conventions = "CF-2.0" ;
}
```

- Example CDL on right demonstrates a proposed method to store multiple geometry types with associated time-varying variables in a single file.
- This example uses variable length arrays for storing geometry coordinates.
- The geometry (instance) dimension “geom_dimension” on coordinate index variables identifies the shared dimension for data variables.
- Using this approach, it is possible to store geometries and data with different instance counts in the same NetCDF file or “group”.

Roadmap

- Ultimate Goals:
  - Inclusion in the CF 1.8 specification using NetCDF-3 contiguous ragged support.
  - Support variable-length (ragged) arrays in the CF 2.0 specification.
- Draft specification for review by the CF committee in early 2017.
- Continue building a coalition of interested parties to contribute to and/or support the simple geometries effort.

How to Get Involved

Please join us or follow our progress. The specification and reference implementations are openly developed GitHub. Community feedback is very valuable and all input is appreciated.

- Hosted on GitHub: https://github.com/bekozi/netCDF-CF-simple-geometry.
  - Draft Specification: https://github.com/bekozi/netCDF-CF-simple-geometry/wiki.
  - Python Reference Implementation: https://github.com/bekozi/netCDF-CF-simple-geometry/tree/master/src/python.
  - R Reference Implementation: https://github.com/dblodgett-usgs/NCDFSG.