Advanced Visualization with ParaView

Generic Data Set API

David Thompson
Sandia National Laboratories

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

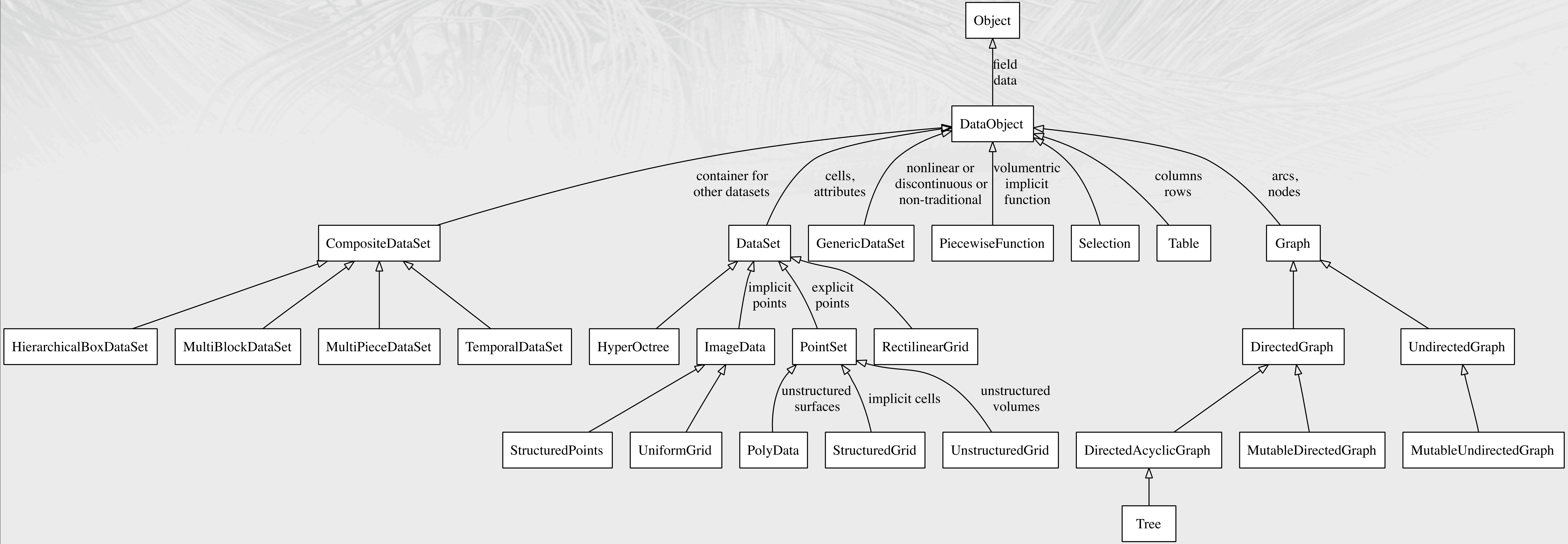
Outline

- Example code will be provided. 40 minutes is not enough time to provide a detailed implementation, so one is provided for our running example: edge/face elements.

- Introduction
- Is GenericDataSet a match for your application?
- Class structure
- Notes for implementation
- Adding to ParaView

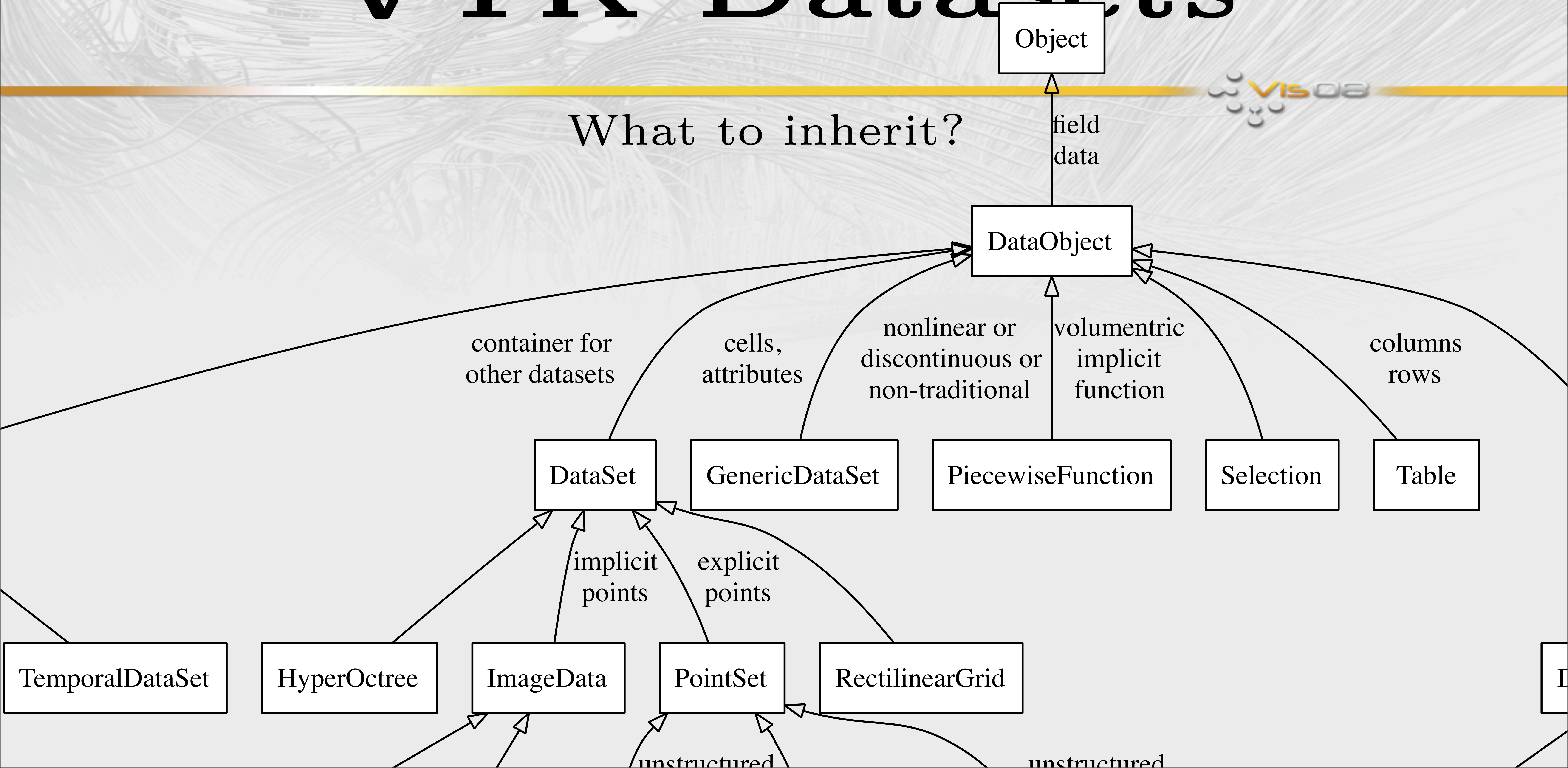
VTK Datasets

What to inherit?



VTK Datasets

What to inherit?



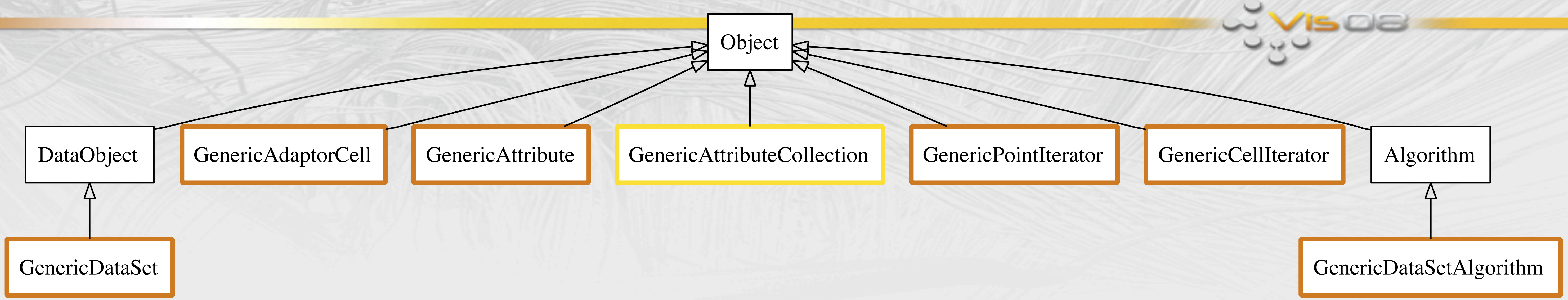
Introduction

- Allow proprietary solvers to present simulation results
- Provide a way for novel PDE solvers to present their results
  - Many new solvers use esoteric cell types (higher order, edge/face elements, X-FEM elements with discontinuities, etc.)
  - Assumptions that VTK algorithms make do not apply
    - Discontinuities at shared boundaries or cell interiors
    - Maxima and minima interior to cell or its boundaries
    - Attributes interpolation may be dependent on geometry
- Purpose is reduction for visualization, not significant analysis

Assumptions

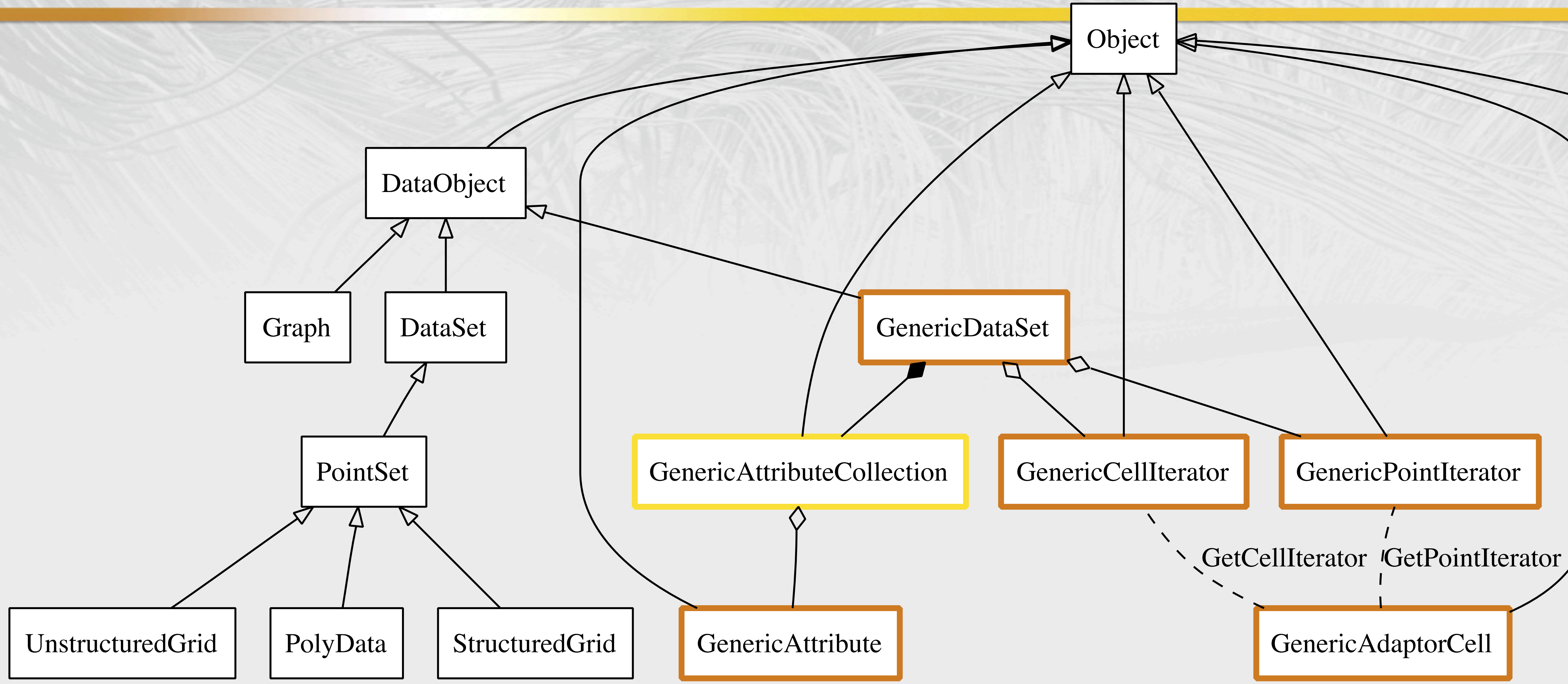
- When subclassing the generic dataset API,
  - Generic datasets are read-only
  - Cell type may not fully specify interpolant the way vtkCell does. Example: p-refinement needs shape, order (r,s,t), and polynomial basis
  - New cell types must provide traditional vis. operations (interpolation, location, line intersection, clip, contour, ...)
  - Filters on generic algorithms may not have access to entire mesh definition; you may need to write mesh-specific filters.
- Facilities exist for approximating a dataset with an unstructured grid

Class Hierarchy



- **GenericDataSet**: Hold mesh data in a compact, private format
- **GenericAdaptorCell**: Provide public access to one cell's mesh data
- **GenericAttribute**: Access and interpolate field data
- **GenericPointIterator** (**GenericCellIterator**): Ordered access to points (cells) within a mesh

Class Hierarchy

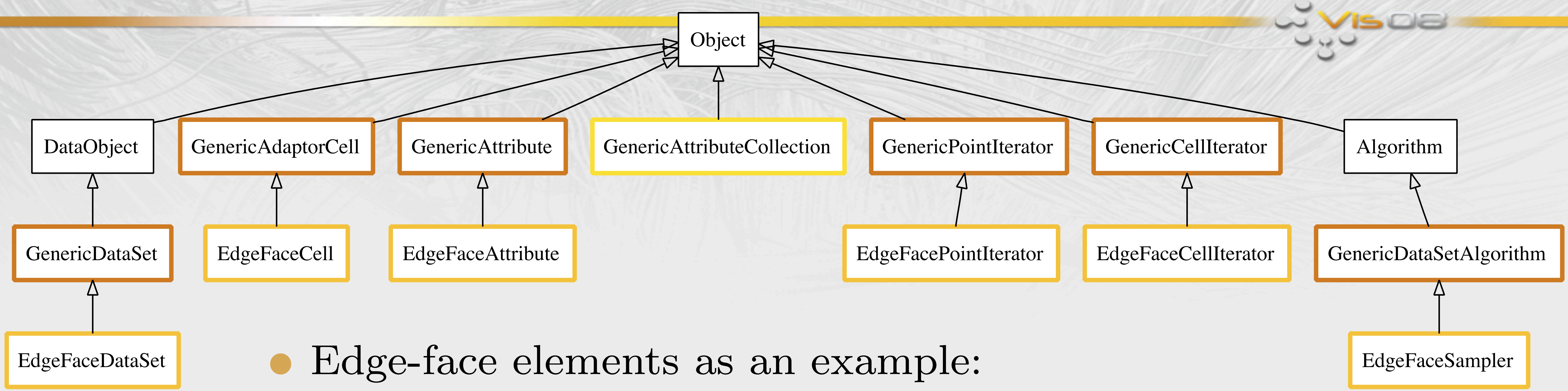


Running Example

- Edge/face elements (actually point←edge←face←cell elements)
- Each set of basis functions is dual to discrete boundary operator and obtained by applying div, grad, or curl to previous entry in sequence.
- Scalars on cells and points yield scalar defined on cell
- Scalars on edges and faces yield vectors defined on cell
- Values not cell or point centered, not isoparametric

  ⇒ GenericDataSet is for us.✓

Class Hierarchy



- Edge-face elements as an example:
  - Attributes are *boundary*-centered (edges or faces).
  - Scalar values stored on boundary yield vector fields.
  - We must implement the classes in green.

Attributes

- This class should hold necessary (but not always sufficient) information for interpolating values on cells;
- Because interpolant may be tied to cell type, an attribute may not be usable without a pointer to the dataset.
  Example: p-refined elements need basis, order, & coefficients.
- **Centering** may be *Point*, *Cell*, or *Boundary*. This is not meant to be exhaustive, only to aid filters that must work on all types of GenericDataSets. You may need to store add'l info.

Cell

- Cell is responsible for interpolation of attributes, including geometry.
  - Think of cell as an iterator over the mesh; it need not store any attribute values.
  - However, all boundaries (d=0,1,2) of cells in mesh must be represented as cells in their own right.
- Contouring (of attributes and geometry), intersection, derivatives, bounds, evaluation, and inverse lookup must all be provided by your implementation.

Cell (cont.)

- Facilities for tessellating cells are provided in order to generate primitives for rendering because the API is geared towards nonlinear geometry + fields and video HW is not.
- The GenericCellTessellator adaptively samples geometry and/or fields;
- Because the mathematics of novel PDE solvers varies so much, you are responsible for providing an initial tessellation that captures all salient feature topology. Override the provided `vtkGenericAdaptorCell::Tessellate()` method to do this.

Cell (cont.)

- Degree of freedom (DOF): a value used to characterize a cell's shape or attributes that is not a geometric coordinate. **Example**: For spectral finite elements, this is the magnitude associated with a given mode shape.
- With higher-order elements, an arbitrary number of DOF may be associated with a cell.
- DOFs may be grouped together by how they are shared among neighboring cells. Each group of DOF values is called a DOF node and there may be one for each boundary of a cell.

Running Example

- Edge/face elements have a DOF node for each edge and face of a cell.
- Since we are only considering hexahedra, each element will have $12 + 6 = 18$ DOF nodes.
- The total number of boundaries in the mesh determine the number of values each element must store.
  Example: 2 hexahedra sharing a face and 4 edges will have
  - Edge attributes specified with 20 values.
  - Face attributes specified with 11 values.

Cell Iterators

- You must implement an iterator that can traverse
  - each cell in the mesh,
  - each cell boundary in the mesh,
  - each cell boundary on the mesh boundary.
- Users can request cells/boundaries of a given dimension.
  - Note that when asked to traverse mesh boundaries of dim 2, no surface cells in the mesh should be included.
  - Likewise, when traversing mesh boundaries of dim 1, do not include edges that appear as cells in the mesh directly.

Point Iterators

- The `GenericDataSet::NewPointIterator()` should prepare an iterator that will visit all mesh points.
- But the CellIterator may need to traverse
  - the points associated with a single cell, or
  - the points associated with a single boundary of a single cell.
- You may implement separate subclasses for each type of traversal, but usually it's simplest to put all of these traversal rules into one class.

DataSet

- The most painful methods in this class are
  - `FindPoint()` and `FindCell()`
  - Cannot use VTK's point locators because they require data objects which inherit `vtkDataSet`. `vtkGenericDataSet` inherits `vtkDataObject` because `vtkDataSet` would require the mesh to present cells using `vtkCell`.

ParaView Plugin

- Plugin must include your reader and (for ParaView 3.4 or newer) any filters from the vtkGenericFiltering library you want to expose.
- Need GUI and ServerManager XML files. From CMake:

```
ADD_PARAVIEW_PLUGIN( EdgeFaceElements "1.0"
      SERVER_MANAGER_XML EdgeFaceServerManager.xml
      GUI_RESOURCE_FILES EdgeFaceUserInterface.xml
      SERVER_MANAGER_SOURCES ${EDGEFACE_SRCS}
      )
```

- The "1.0" is a version number and `EDGEFACE_SRCS` is the list of C++ files implementing generic dataset API subclasses.

ParaView Plugin

- GUI XML is trivial for most filters. See the example.
- ServerManager XML:

```
<ServerManagerConfiguration>
  <ProxyGroup name="sources">
    <SourceProxy name="EdgeFaceReader"
     class="vtkEdgeFaceReader" label="Edge/face reader">
      <Documentation short_help="Read edge/face meshes."
       long_help="...">
        The edge/face reader ...
      </Documentation>
    </SourceProxy>
  </ProxyGroup> <!-- any filters go here -->
</ServerManagerConfiguration>
```

Conclusion

- Frequently use SafeDownCast() in order to access methods specific to your implementation from another class. Example: Cell's InterpolateTuple() to access Attribute.
- Remember the point is to generate primitives that can be rendered or analysis that can be displayed; expect to use the tessellator, write a custom filter, or write a custom mapper.