SAND2015-7296PE

Add Cool Visualizations Here

VTK-m Overview

VTK-m Code Sprint

September 1-2, 2015

Kenneth Moreland

Sandia National Laboratories

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2015-XXXX PE

Acknowledgements

- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Numbers 10-014707, 12-015215, and 14-017566.

Motivation

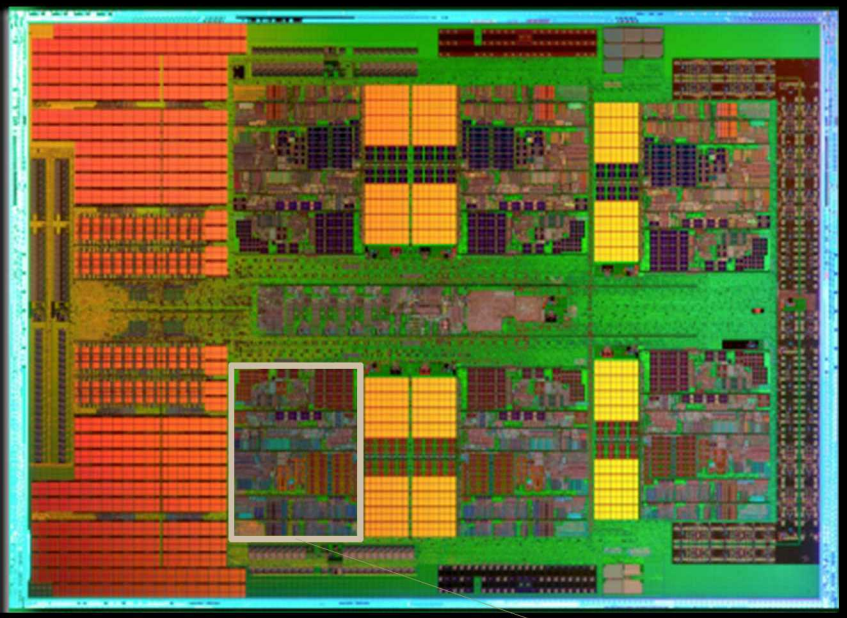
Supercomputers!

- A clear trend in supercomputing is ever increasing parallelism
- Clock increases are long gone
  - "The Free Lunch Is Over" (Herb Sutter)

	Jaguar – XT5	**Titan – XK7**	**Exascale***
Cores	224,256	299,008 and 18,688 gpu	1 billion
Concurrency	224,256 way	70 – 500 million way	10 – 100 billion way
Memory	300 Terabytes	700 Terabytes	128 Petabytes

*Source: Scientific Discovery at the Exascale, Ahern, Shoshani, Ma, et al.

“Everybody who learns concurrency thinks they understand it, ends up finding mysterious races they thought weren’t possible, and discovers that they didn’t actually understand it yet after all.” Herb Sutter

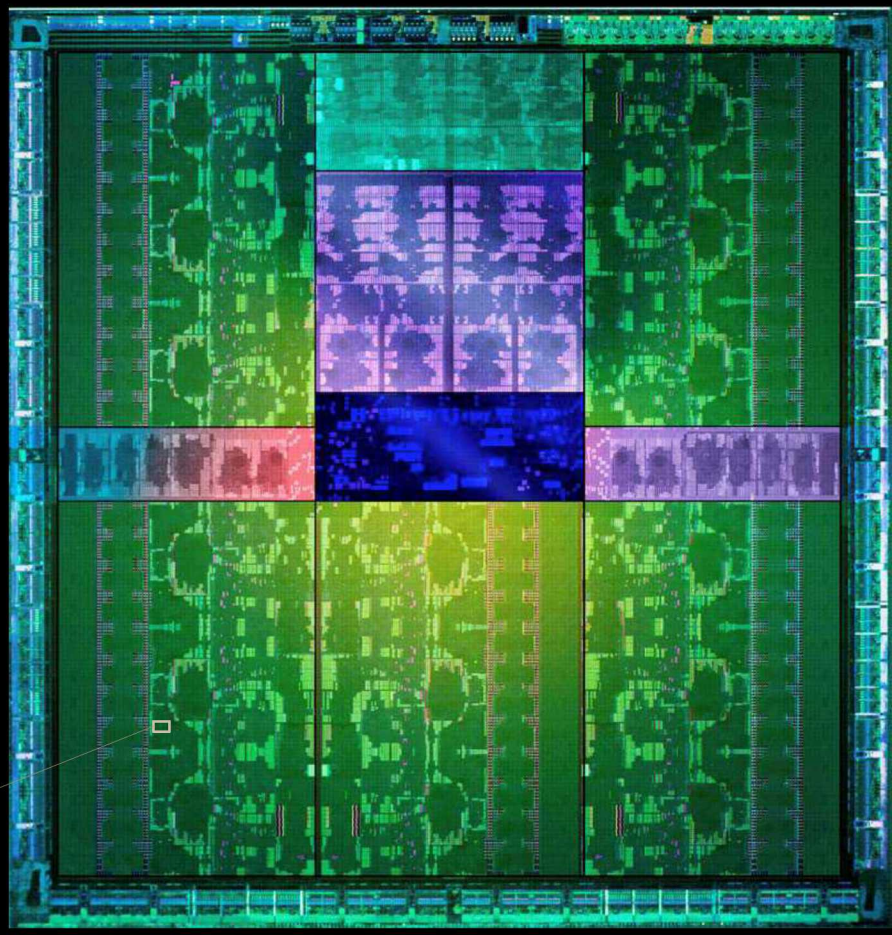


AMD x86

Full x86 Core
+ Associated Cache
6 cores per die
MPI-Only feasible

1 x86 core

1 Kepler "core"



NVIDIA GPU

2,880 cores collected in 15 SMX
Shared PC, Cache, Mem Fetches
Reduced control logic
MPI-Only not feasible

Getting Started

Prerequisites

- Always required:
  - git
  - CMake (2.10 or newer)
  - Boost 1.48.0 (or newer)
  - Linux, Mac OS X, or MSVC
- For CUDA backend:
  - CUDA Toolkit 7+
  - Thrust (comes with CUDA)
- For Intel Threading Building Blocks backend:
  - TBB library

Getting VTK-m

- http://m.vtk.org → Building VTK-m
- Clone from the git repository
  - https://gitlab.kitware.com/vtk/vtk-m.git

```
git clone http://gitlab.kitware.com/vtk/vtk-m.git
mkdir vtk-m-build
cd vtk-m-build
ccmake ../vtk-m
make
ctest
```

Configuring VTK-m

- http://m.vtk.org → Building VTK-m
- Create a build directory
- Run ccmake (or cmake-gui) pointing back to source directory

```
git clone http://gitlab.kitware.com/vtk/vtk-m.git
mkdir vtk-m-build
cd vtk-m-build
ccmake ../vtk-m
```

Important Configuration Parameters

Variable	Description
VTKm_ENABLE_CUDA	Enable CUDA backend. Requires CUDA Toolkit.
VTKm_ENABLE_OPENMP	Enable OpenMP backend. Requires OpenMP compiler support (not Clang). (Coming soon.)
VTKm_ENABLE_TBB	Enable Intel Threading Building Blocks backend. Requires the TBB library.
VTKm_ENABLE_TESTING	Turn on header, unit, and benchmark tests.
VTKm_ENABLE_BENCHMARKS	Turn on additional timing tests.
VTKm_USE_64BIT_IDS	Enable 64bit index support. Older CUDA cards might not support 64 bit integers.
VTKm_USE_DOUBLE_PRECISION	Precision to use in floating point numbers when no other precision can be inferred. Older CUDA cards might not support 64 bit floats.
CMAKE_BUILD_TYPE	Debug, RelWithDebInfo, or Release
CMAKE_INSTALL_PREFIX	Location to install headers

Building VTK-m

- http://m.vtk.org → Building VTK-m
- Run make (or use your favorite IDE)
- Run tests (“make test” or “ctest”)
- Parallel builds (-j flag) work, too
  - Good idea to use them as building VTK-m can take a while

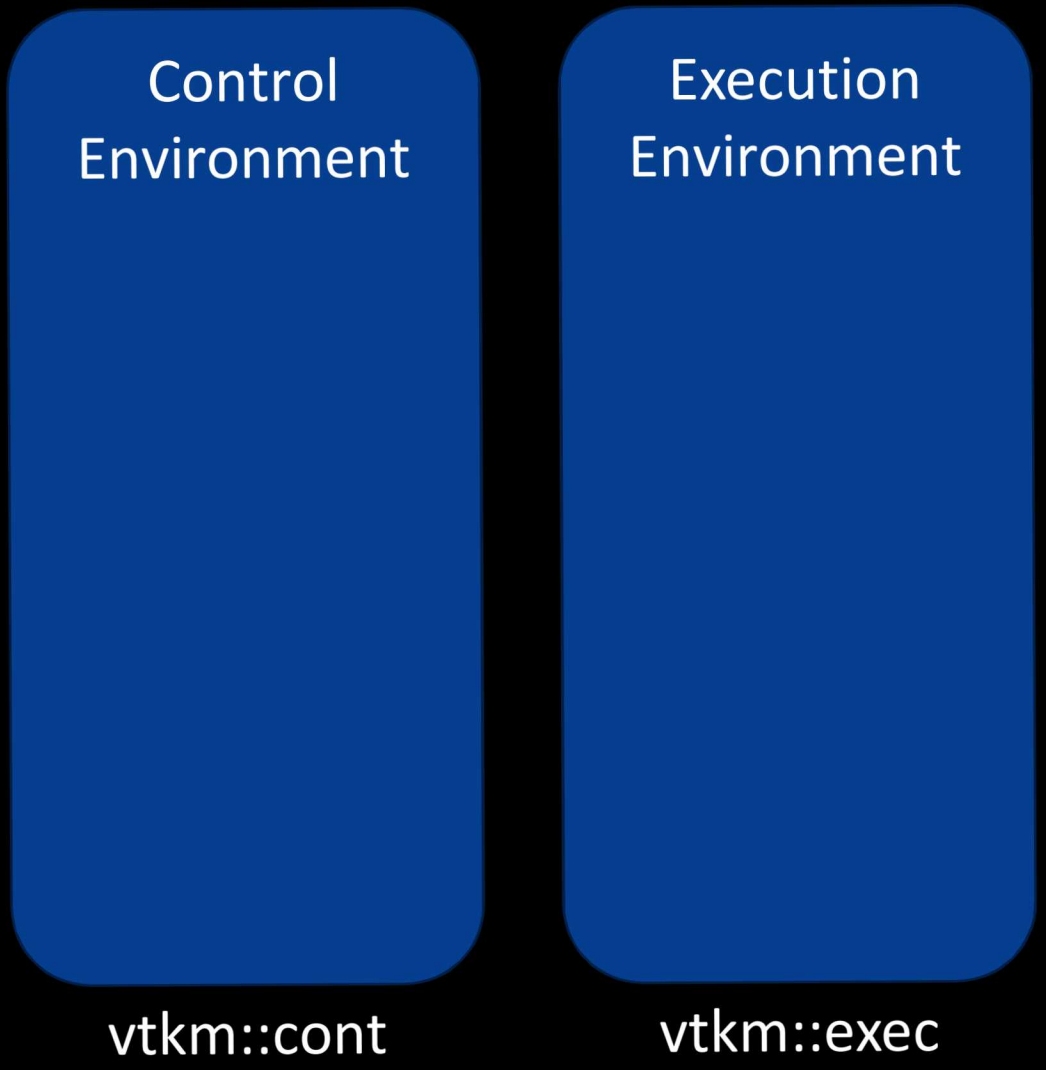
```
git clone http://gitlab.kitware.com/vtk/vtk-m.git
mkdir vtk-m-build
cd vtk-m-build
ccmake ../vtk-m
make
ctest
```

More Information

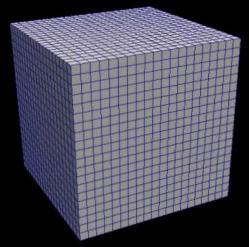
- We know, documentation is sparse
- http://m.vtk.org
- A user's guide is on its way. We are also working on a textbook.
- Doxygen:

System Overview

VTK-m Framework



VTK-m Framework



Control Environment

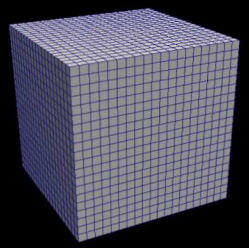
Data Set
Array Handle
Invoke

vtkm::cont

Execution Environment

vtkm::exec

VTK-m Framework



Control Environment

Data Set
Array Handle
Invoke

Execution Environment

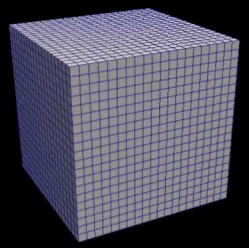
Cell Operations
Field Operations
Basic Math
Make Cells

Worklet

vtkm::cont

vtkm::exec

VTK-m Framework



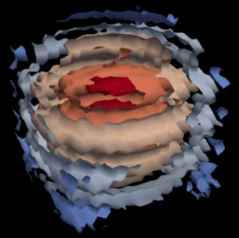
Control Environment

Data Set
Array Handle
Invoke

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

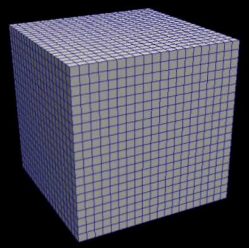
Worklet



vtkm::cont

vtkm::exec

VTK-m Framework



Control Environment

Data Set
Array Handle
Invoke

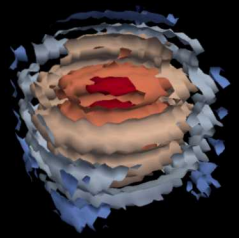
Device Adapter

Allocate
Transfer
Schedule
Sort
...

Execution Environment

Cell Operations
Field Operations
Basic Math
Make Cells

Worklet

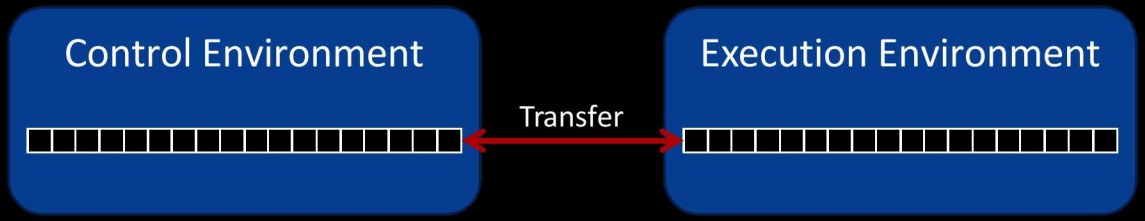


vtkm::cont

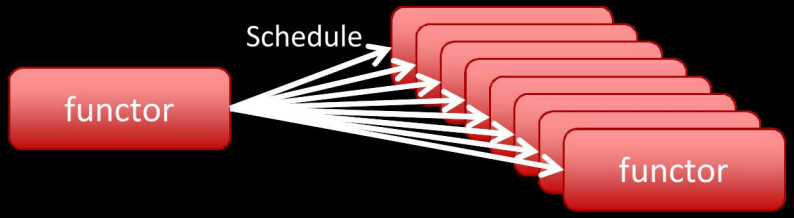
vtkm::exec

Device Adapter Contents

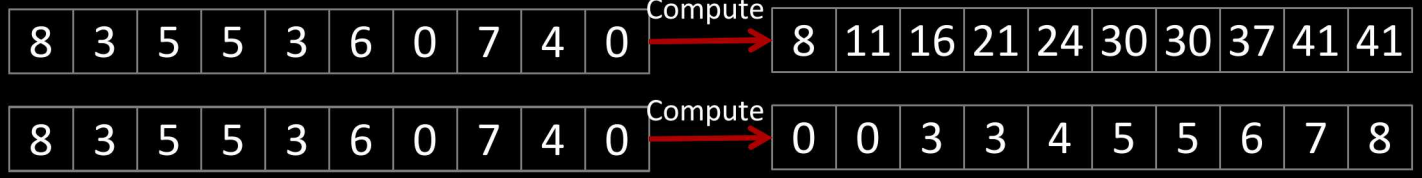
- Tag (`struct DeviceAdapterFoo {  };`)
- Execution Array Manager



- Schedule



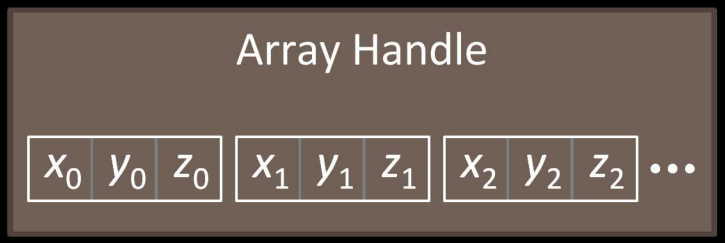
- Scan
- Sort



- Other Support algorithms
  - Stream compact, copy, parallel find, unique

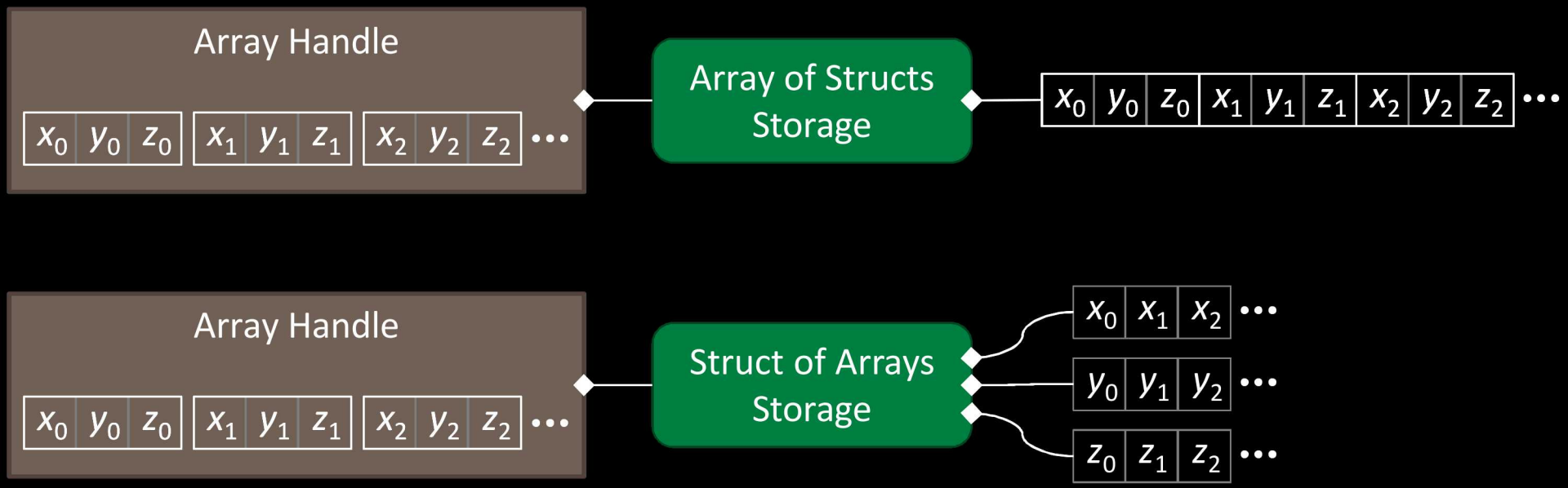
Defining Data

Array Handle

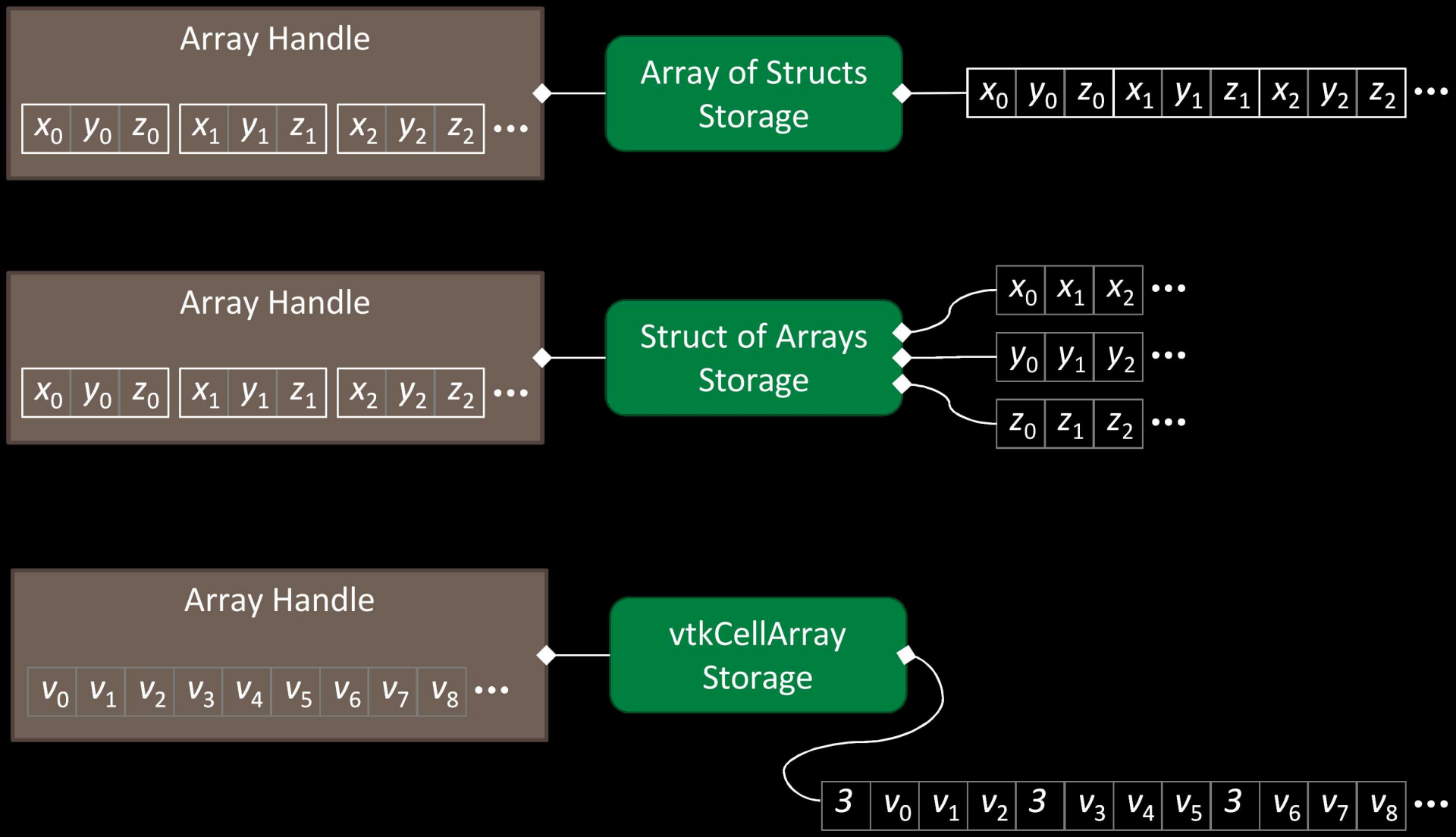


Array Handle Storage

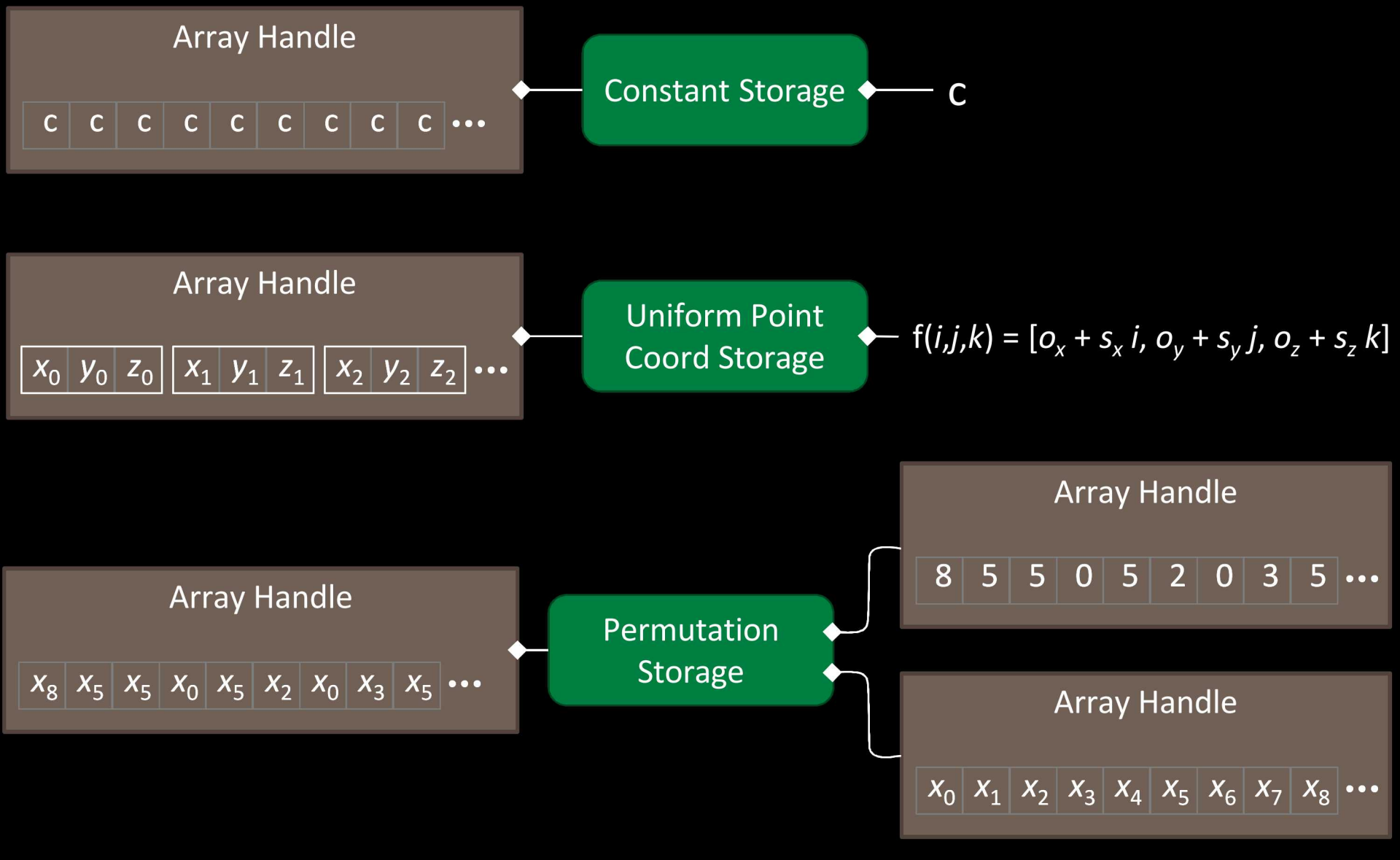
Array Handle Storage



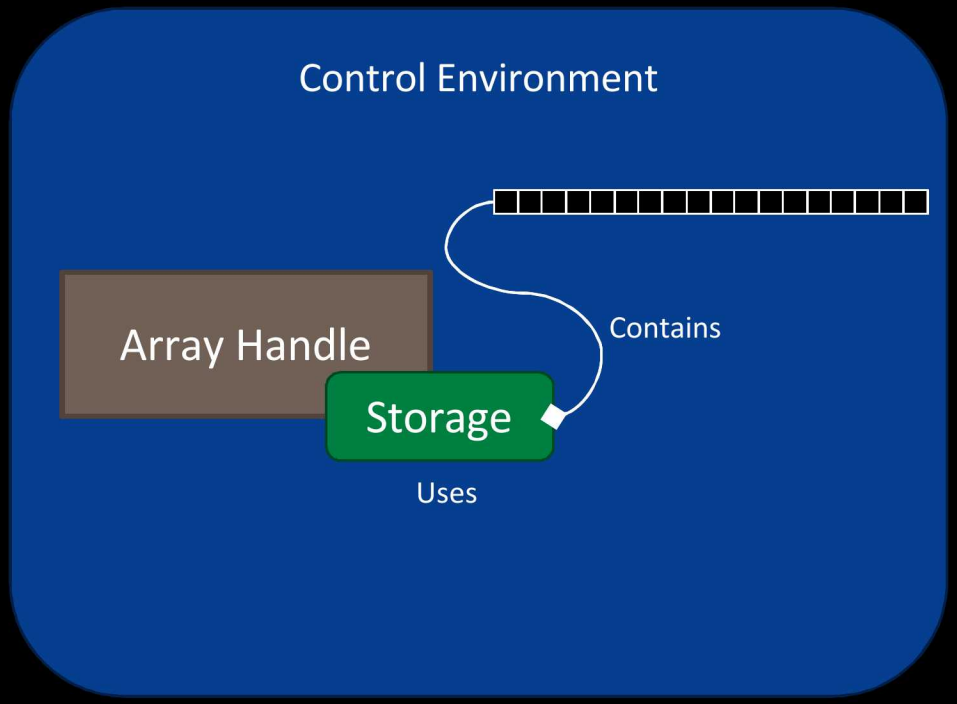
Array Handle Storage



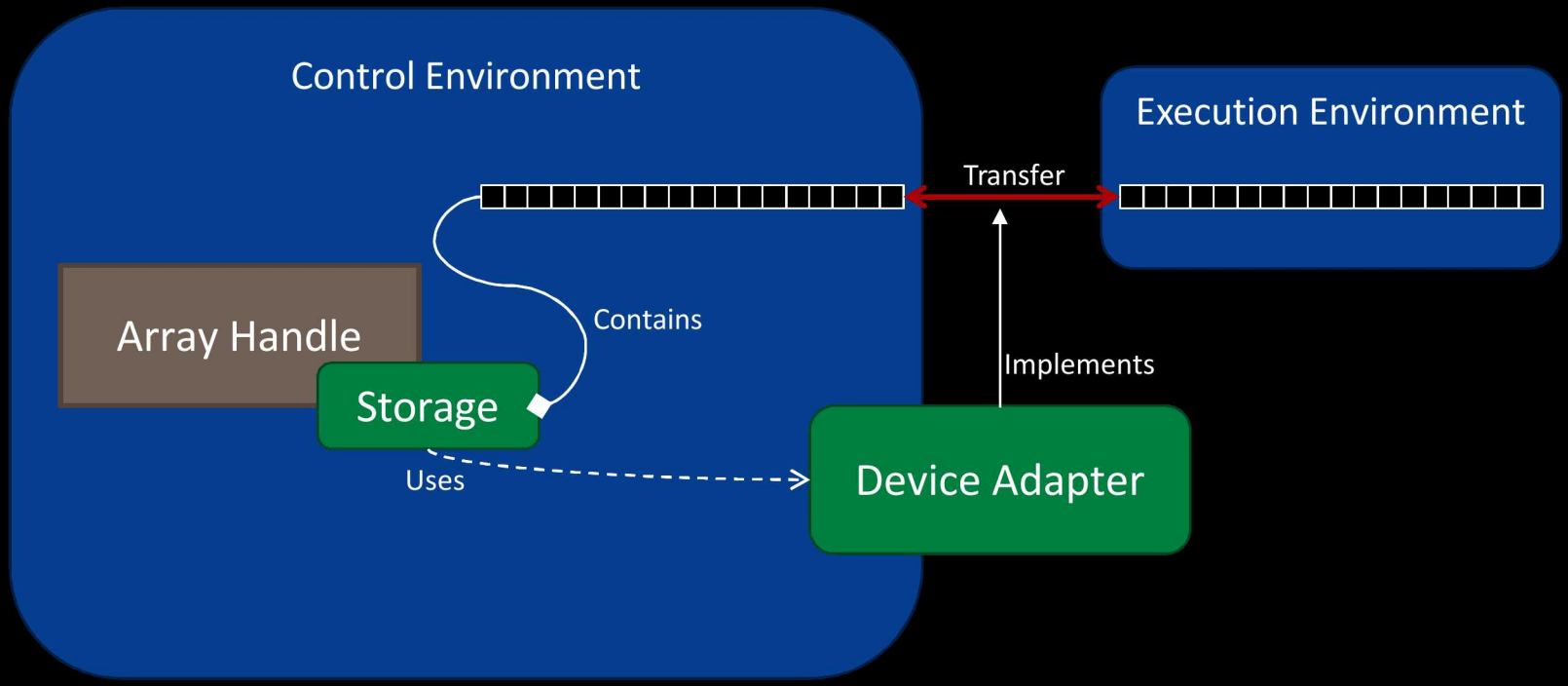
Fancy Array Handles



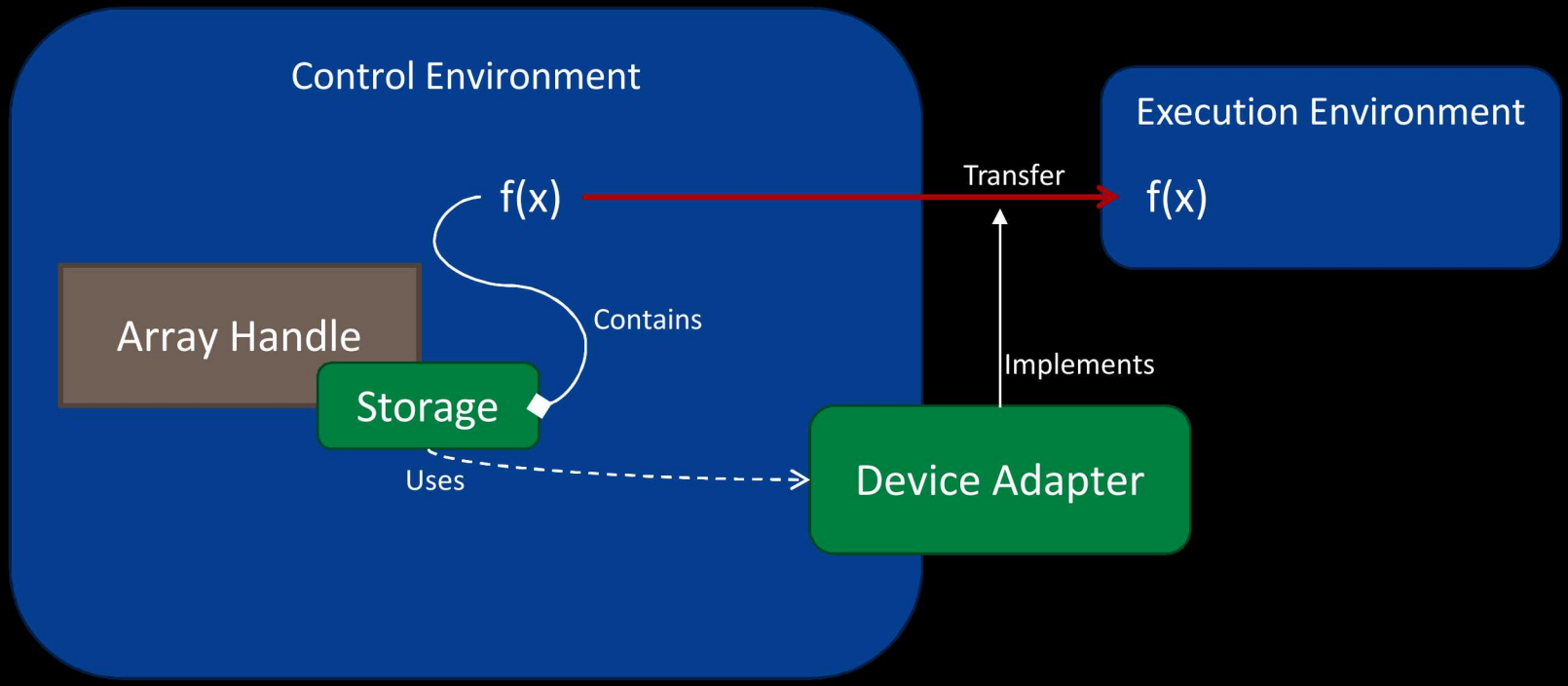
Array Handle Resource Management



Array Handle Resource Management



Array Handle Resource Management



ArrayHandle

- `vtkm::cont::ArrayHandle<type>` manages an "array" of data
  - Acts like a reference-counted smart pointer to an array
  - Manages transfer of data between control and execution
  - Can allocate data for output
- Relevant methods
  - `GetNumberOfValues()`
  - `GetPortalConstControl()`
  - `ReleaseResources()`, `ReleaseResourcesExecution()`
- Functions to create an ArrayHandle
  - `vtkm::cont::make_ArrayHandle(const T*array,vtkm::Id size)`
  - `vtkm::cont::make_ArrayHandle(const std::vector<T>&vector)`
  - Both of these do a shallow (reference) copy.
    - Do not let the original array be deleted or vector to go out of scope!

Fancy Array Handles

- ArrayHandleCompositeVector: Zips components of source arrays to compose vector values
- ArrayHandleConstant: An array with a single constant value
- ArrayHandleCounting: An array that starts at an index and counts up
- ArrayHandleImplicit: Each entry of the array is the value returned from the provided functor
- ArrayHandlePermutation: Rearranges the entries in one ArrayHandle by the indices of another ArrayHandle
- ArrayHandleTransform: Modifies the values of one ArrayHandle by feeding them through a provided functor
- ArrayHandleUniformPointCoordinates: Defines the point coordinates from a uniform rectilinear grid
- ArrayHandleZip: An array of Pair with values coming from two provided arrays

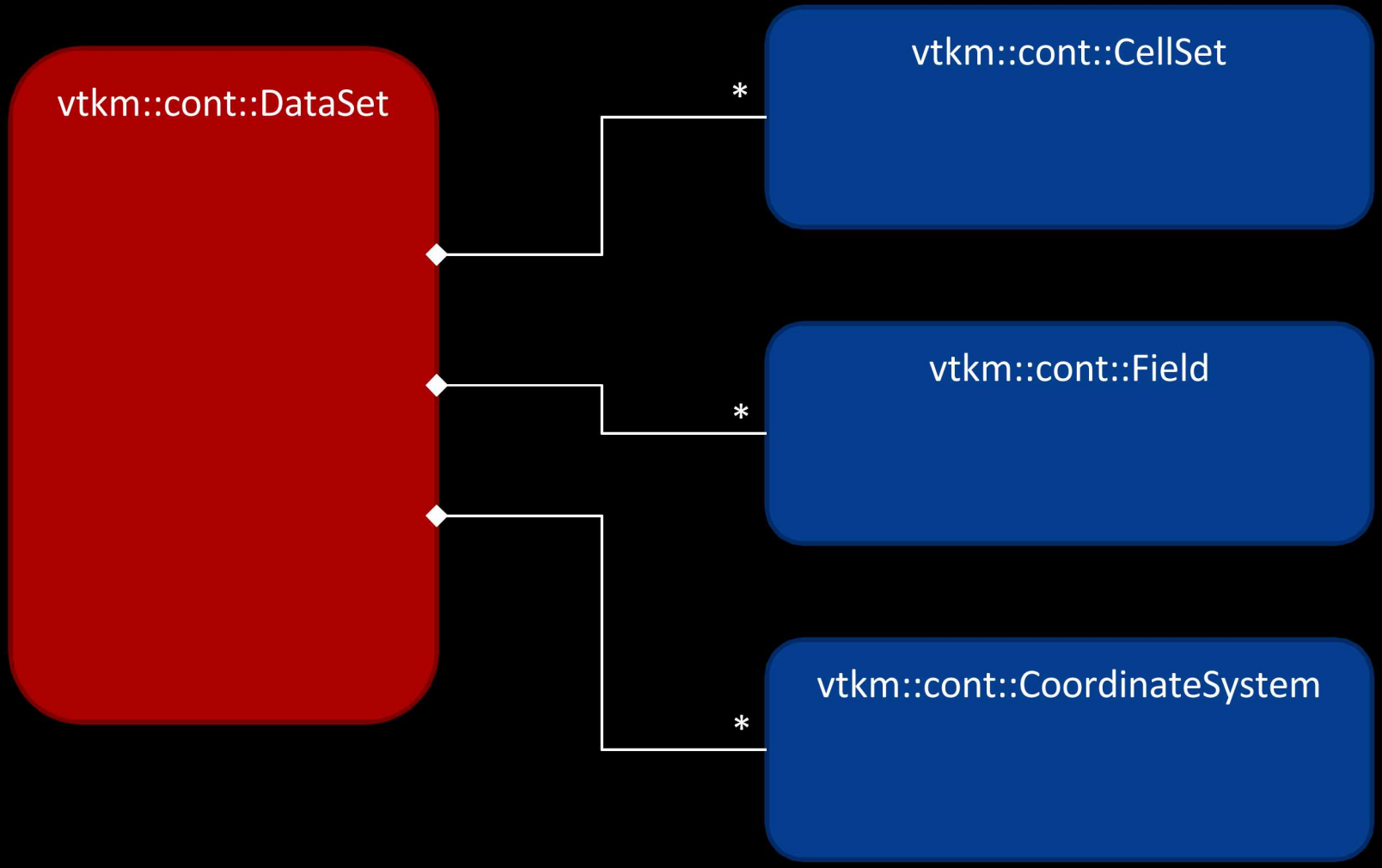
Other Important ArrayHandle Features We're Skipping

- Storage template parameter
  - Selects array layout for zero-copy semantics
  - Supports implicit and derived arrays (all those fancy array handles)
- Generic array interface for data through an ArrayPortal
  - In principle like an STL iterator, but simpler
  - Type of object returned from `GetPortalConstControl`
  - Can exist in execution environment (depends on definition)

DynamicArrayHandle

- DynamicArrayHandle is a magic untyped reference to an ArrayHandle
- Statically holds a list of potential types and storages the contained array might have
  - Can be changed with ResetTypeList and ResetStorageList
  - Changing these lists requires creating a new object
- Parts of VTK-m will automatically staticly cast a DynamicArrayHandle as necessary
  - Requires the actual type to be in the list of potential types

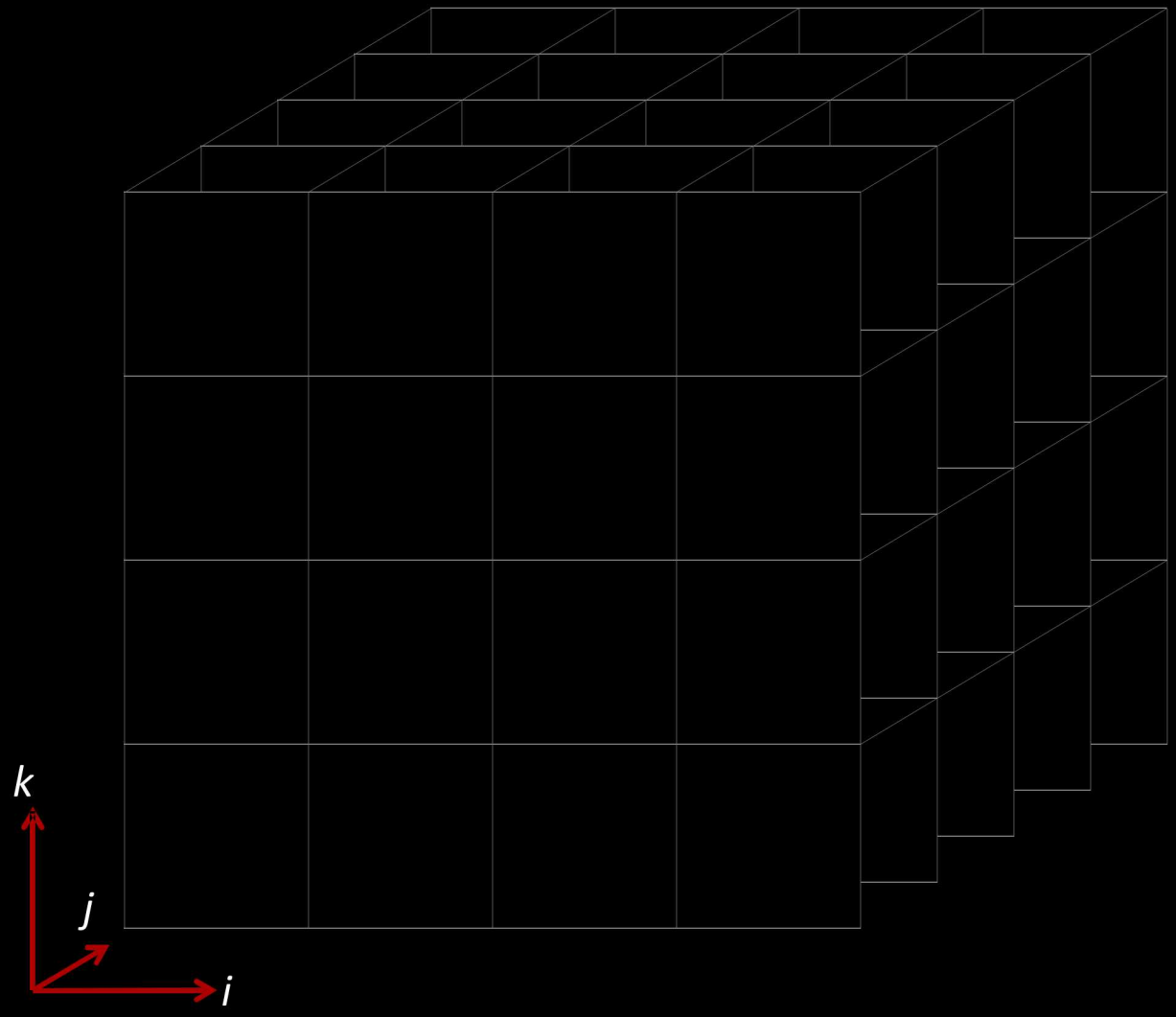
Data Model


vtkm::cont::DataSet
*
vtkm::cont::CellSet
*
vtkm::cont::Field
*
vtkm::cont::CoordinateSystem

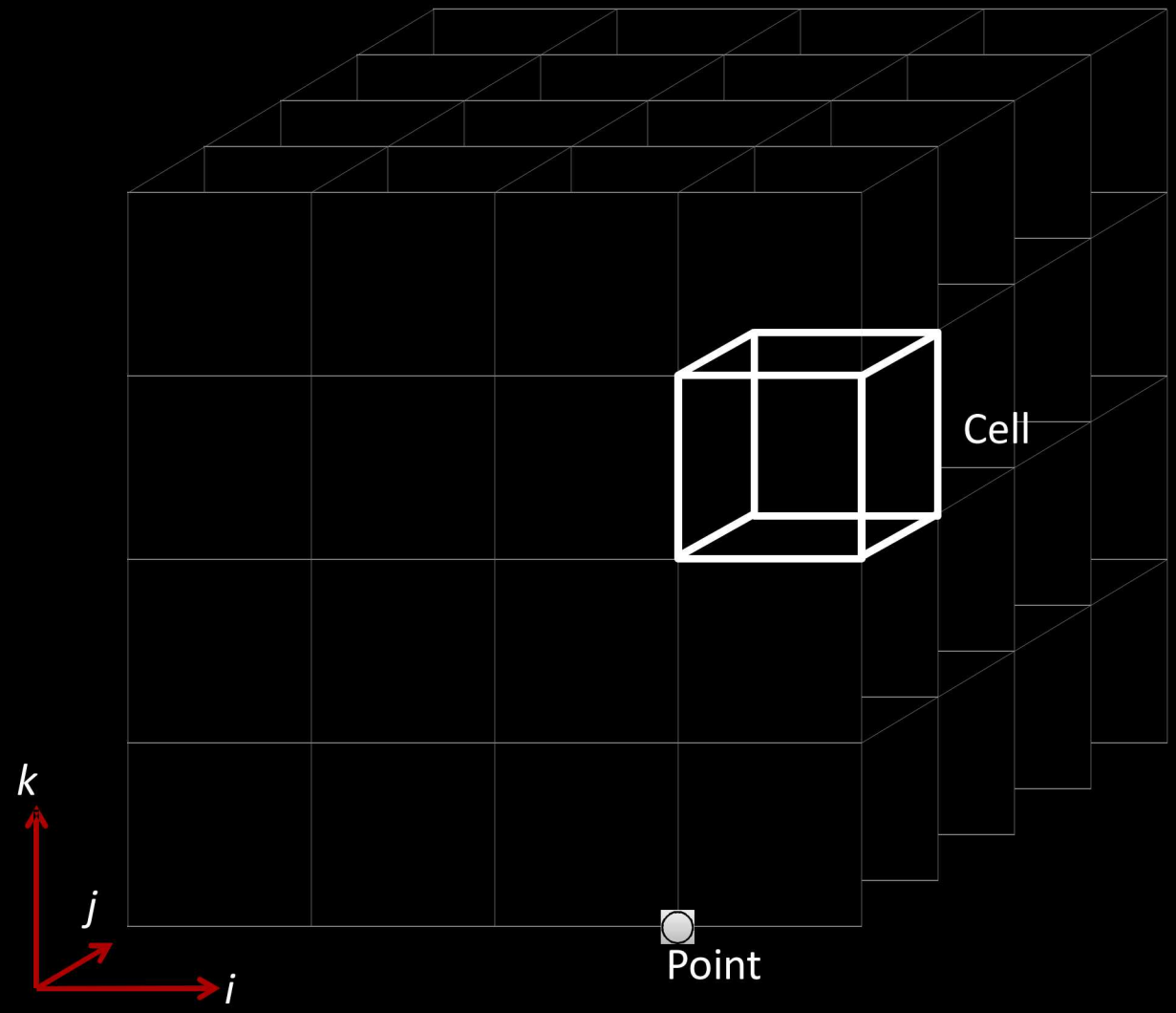
A DataSet Has

- 1 or more CellSet
  - Defines the connectivity of the cells
  - Examples include a regular grid of cells or explicit connection indices
- 0 or more Field
  - Holds an ArrayHandle containing field values
  - Field also has metadata such as the name, the topology association (point, cell, face, etc), and which cell set the field is attached to
- 0 or more CoordinateSystem
  - Really just a Field with a special meaning
  - Contains helpful features specific to common coordinate systems

Structured Cell Set



Structured Cell Set



Example: Making a Structured Grid


```
vtkm::cont::DataSet dataSet;

const vtkm::Id nVerts = 18;
const vtkm::Id3 dimensions(3, 2, 3);

// Build cell set
vtkm::cont::CellSetStructured<3> cellSet("cells");
cellSet.SetPointDimensions(dimensions);
dataSet.AddCellSet(cellSet);

// Make coordinate system
vtkm::cont::ArrayHandleUniformPointCoordinates coordinates(dimensions);
dataSet.AddCoordinateSystem(vtkm::cont::CoordinateSystem("coordinates", 1, coordinates));

// Add point scalar data
vtkm::Float32 vars[nVerts] = {…};
dataSet.AddField(Field("pointvar", 1, vtkm::cont::Field::ASSOC_POINTS, vars, nVerts));

// Add cell scalar
vtkm::Float32 cellvar[4] = {…};
dataSet.AddField(Field("cellvar", 1, vtkm::cont::Field::ASSOC_CELL_SET,
                       "cells", cellvar, 4));
```

Explicit Connectivity Cell Set

Shapes	Num Indices	Map Cell to Connectivity
TETRA	4	0
TETRA	4	4
HEDAHEDRON	8	8
WEDGE	6	16
HEXAHEDRON	8	22
HEXAHEDRON	8	30
TETRA	4	38

Connectivity
0
1
2
3
0
2
1
4
5
6
7
8

Running Worklets

Invoking Worklets

- Each worklet has a type (e.g. Map Field, Generate Topology, etc.)
  - Inherits from a corresponding `vtkm::worklet::Worklet*` class (e.g. `WorkletMapField`, `WorkletMapTopology`, etc.)
- Each worklet type has a corresponding dispatcher
  - Name matches the worklet type (e.g. `vtkm::worklet::DispatcherMapField`)
- All dispatchers take worklet type as first template argument.
- All dispatchers take a worklet instance on its constructor (or construct a new one).
- All dispatcher have an `Invoke` method that runs the worklet on a given set of parameters.

Example: Invoking a Worklet


```
void RunElevation(vtkm::cont::DataSet &dataSet)
{
  vtkm::worklet::PointElevation pointElevationWorklet;
  pointElevationWorklet.SetLowPoint(vtkm::Vec<vtkm::Float64,3>(0.0, 0.0, 0.0);
  pointElevationWorklet.SetHighPoint(vtkm::Vec<vtkm::Float64,3>(1.0, 1.0, 1.0);

  vtkm::worklet::DispatcherMapField<vtkm::worklet::PointElevation>
      dispatcher(pointElevationWorklet);

  vtkm::cont::ArrayHandle<vtkm::Float32> elevationOutput;

  dispatcher.Invoke(dataSet.GetCoordinateSystem().GetData(), elevationOutput);

  dataset.AddField(vtkm::cont::Field("elevation", 1, vtkm::cont::Field::ASSOC_POINTS,
                                     elevationOutput));
}
```

Anatomy of a Worklet

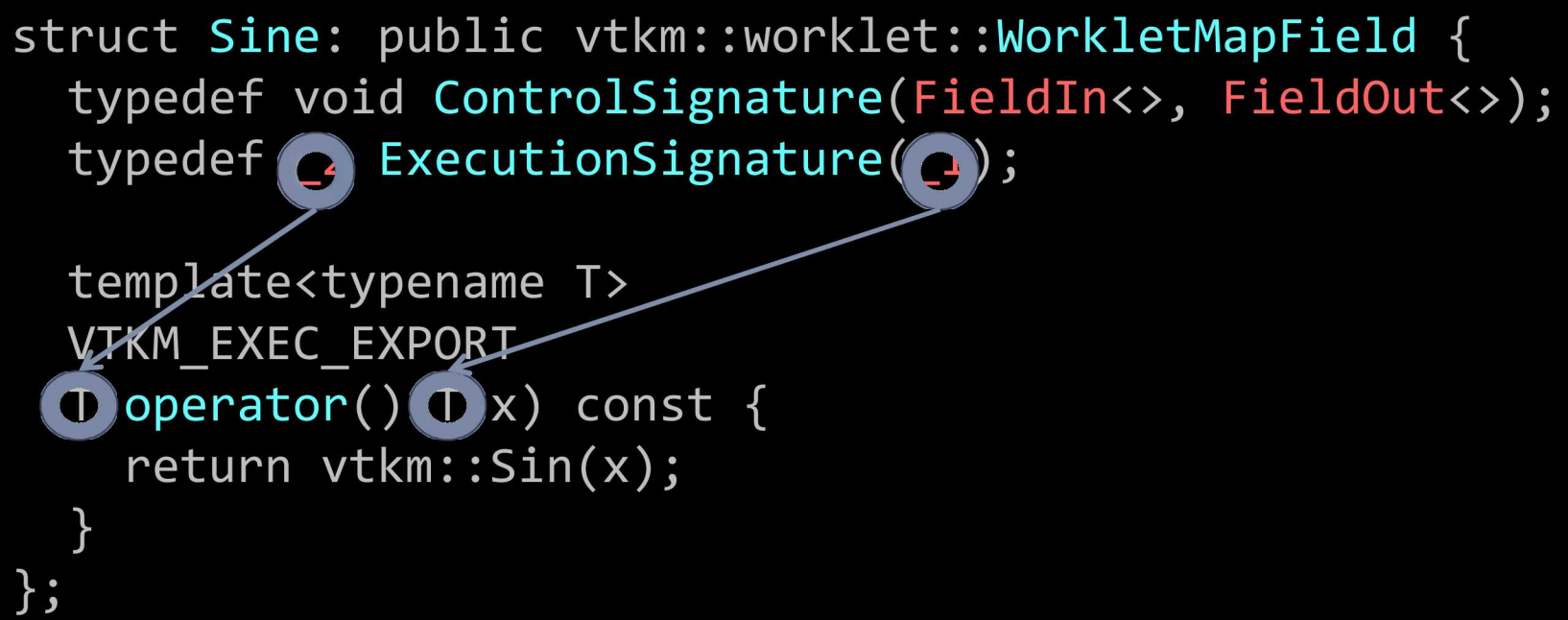
```
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```

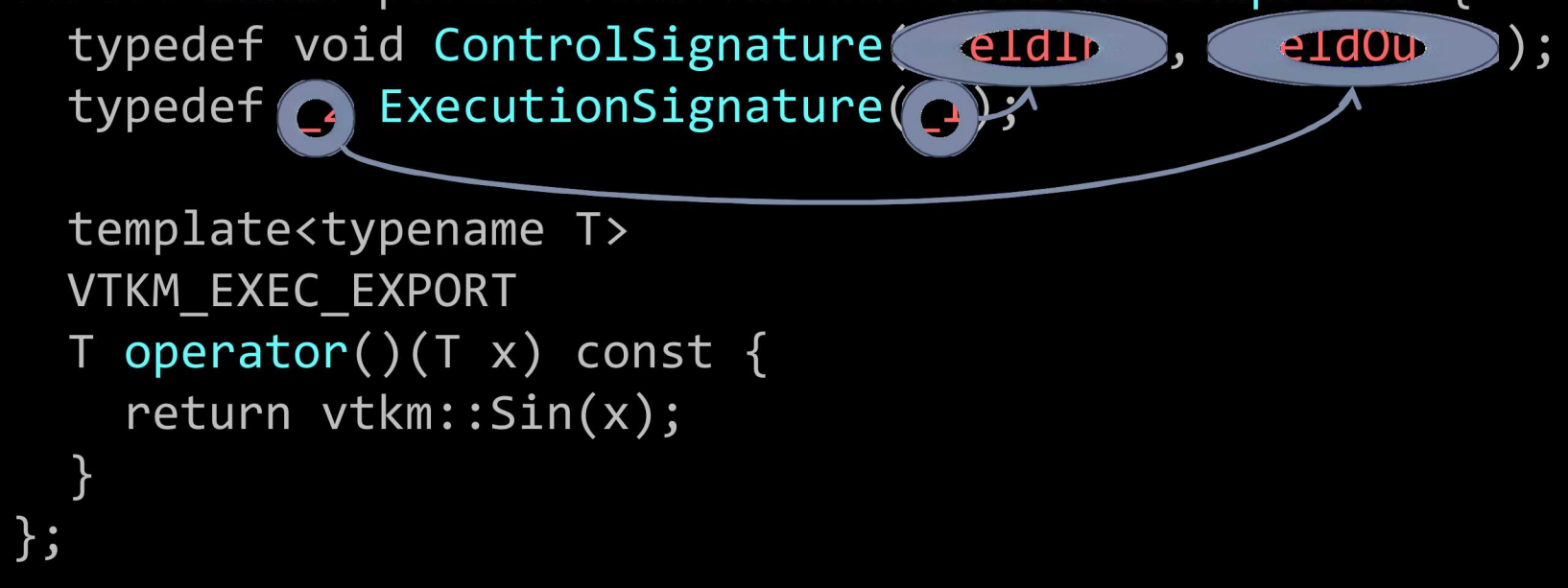
```
template<typename T>
VTKM_EXEC_EXPORT
T operator()(T x) const {
  return vtkm::Sin(x);
}
```

```
struct Sine                                    {


  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```

```
struct Sine: public vtkm::worklet::WorkletMapField {


  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```

```
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```


struct Sine: public vtkm::worklet::WorkletMapField {
typedef void ControlSignature(FieldIn<>, FieldOut<>);
typedef _2 ExecutionSignature(_1);
template<typename T>
VTKM_EXEC_EXPORT
T operator()(T x) const {
return vtkm::Sin(x);
}
};


struct Sine: public vtkm::worklet::WorkletMapField {
typedef void ControlSignature(
);
typedef
ExecutionSignature(
);
template<typename T>
VTKM_EXEC_EXPORT
T operator()(T x) const {
return vtkm::Sin(x);
}
};

```
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```

Execution Environment

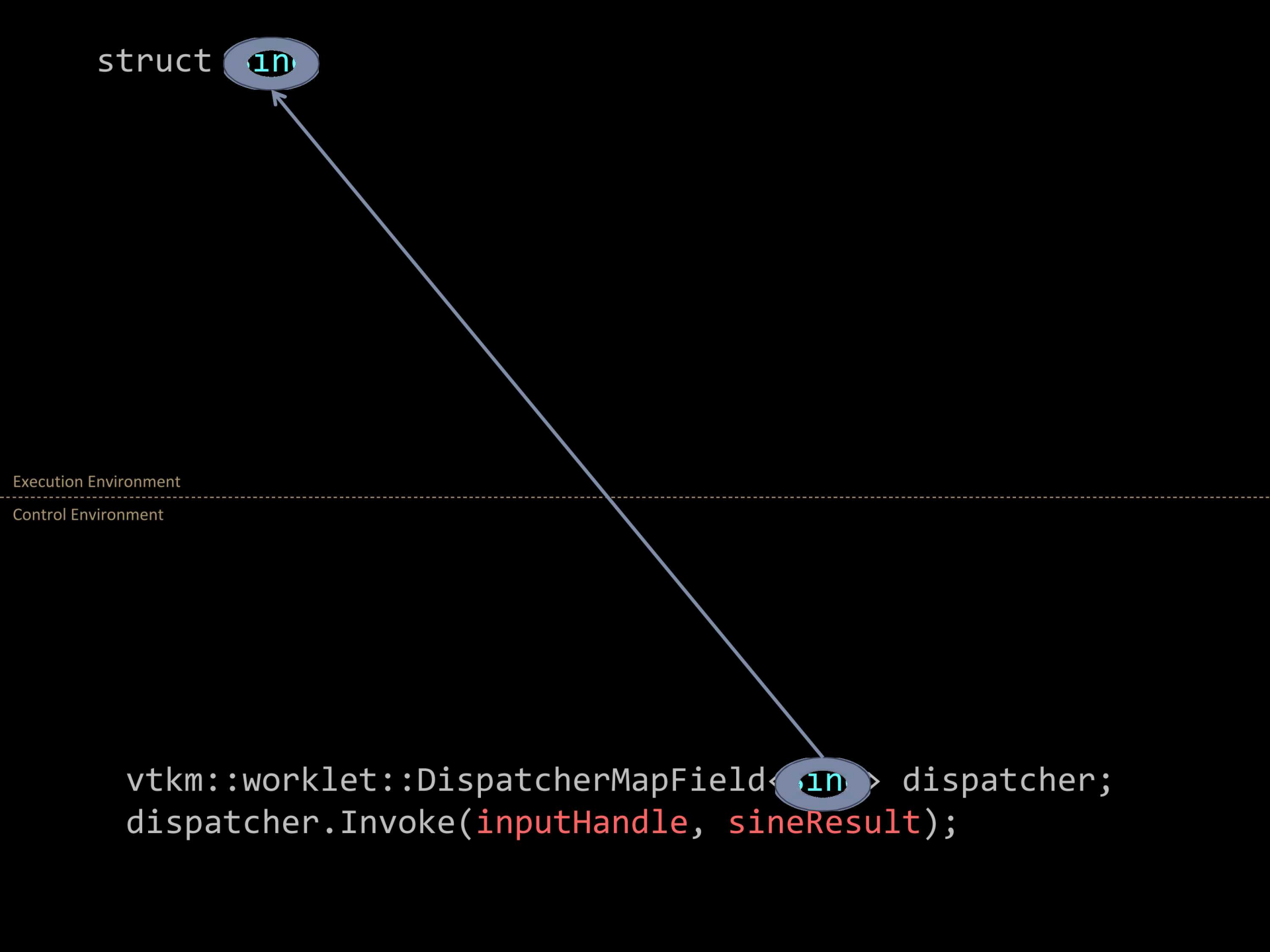
Control Environment

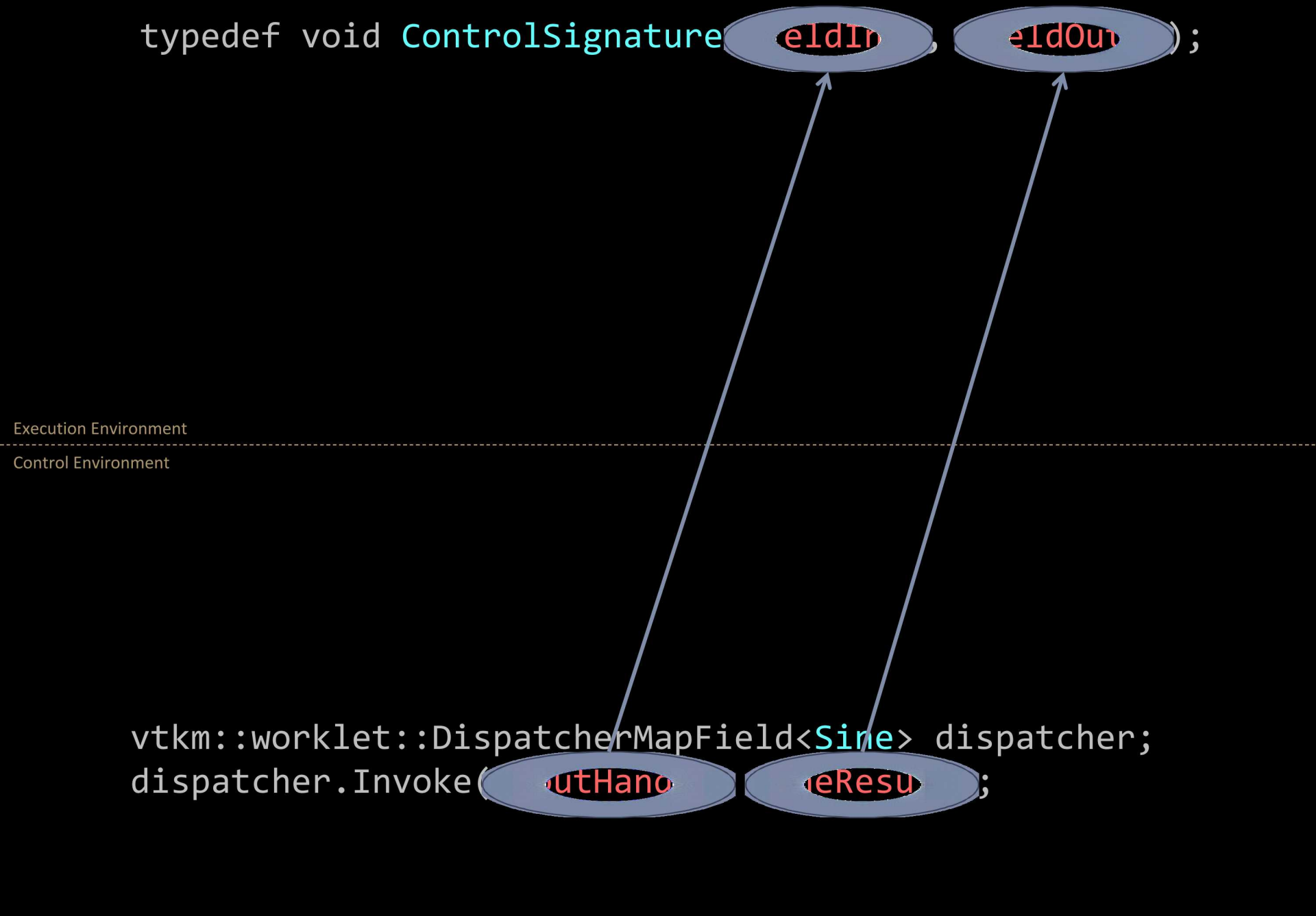
```
vtkm::cont::ArrayHandle<vtkm::Float32> inputHandle =
    vtkm::cont::make_ArrayHandle(input);
vtkm::cont::ArrayHandle<vtkm::Float32> sineResult;

vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
```

Execution Environment

Control Environment

```
vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
```


struct
Execution Environment
Control Environment
vtkm::worklet::DispatcherMapField dispatcher;
dispatcher.Invoke(inputHandle, sineResult);


typedef void ControlSignature
Execution Environment
Control Environment
vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(

```
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT
  T operator()(T x) const {
    return vtkm::Sin(x);
  }
};
```

```
struct Zip2: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(
      FieldIn<vtkm::TypeListTagScalar>,
      FieldIn<vtkm::TypeListTagScalar>,
      FieldOut<vtkm::TypeListTagFieldVec2>);
  typedef void ExecutionSignature(_1, _2, _3);

  typedef<typename T1, typename T2, typename V>
  VTKM_EXEC_EXPORT
  void operator()(T1 x, T2 y, V &result) const {
    result = V(x, y);
  }
};
```

```
struct ImagToPolar: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(
      FieldIn<vtkm::TypeListTagScalar>,
      FieldIn<vtkm::TypeListTagScalar>,
      FieldOut<vtkm::TypeListTagScalar>,
      FieldOut<vtkm::TypeListTagScalar>);
  typedef void ExecutionSignature(_1, _2, _3, _4);

  template<typename RealType,
           typename ImaginaryType,
           typename MagnitudeType,
           typename PhaseType>
  VTKM_EXEC_EXPORT
  void operator()(RealType real,
                  ImaginaryType imag,
                  MagnitudeType &magnitude,
                  PhaseType &phase) const {
    magnitude = vtkm::Sqrt(real*real + imag*imag);
    phase = vtkm::ATan2(imaginary, real);
  }
};
```

```
struct Advect: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(
      FieldIn<vtkm::TypeListTagFieldVec3>,
      FieldIn<vtkm::TypeListTagFieldVec3>,
      FieldIn<vtkm::TypeListTagFieldVec3>,
      FieldOut<vtkm::TypeListTagFieldVec3>,
      FieldOut<vtkm::TypeListTagFieldVec3>,
      FieldOut<vtkm::TypeListTagScalar>,
      FieldOut<vtkm::TypeListTagScalar>);
  typedef void ExecutionSignature(
                  _1, _2, _3, _4, _5, _6, _7);

  template<typename T1, typename T2, ...>
  VTKM_EXEC_EXPORT
  void operator()(T1 startPosition,
                  T2 startVelocity,
                  T3 acceleration,
                  T4 &endPosition,
                  T5 &endVelocity,
                  T6 &rotation,
                  T7 &angularVelocity) const {
```

Creating Worklets

Worklet Types

- `WorkletMapField`: Applies worklet on each value in an array.
- `WorkletMapTopology`: Takes from and to topology elements (e.g. point to cell or cell to point). Applies worklet on each “to” element. Worklet can access field data from both “from” and “to” elements. Can output to “to” elements.
- Many more to come…

Elements of a Worklet

1. Subclass of one of the base worklet types (from previous slide)
2. Typedefs for `ControlSignature` and `ExecutionSignature`
3. A parenthesis operator
   1. Must have VTKM_EXEC_EXPORT
   2. Input parameters are by value or const reference
   3. Output parameters are by reference
   4. The method must be declared const

```
struct ImagToPolar: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<vtkm::TypeListTagScalar>,
                                FieldIn<vtkm::TypeListTagScalar>,
                                FieldOut<vtkm::TypeListTagScalar>,
                                FieldOut<vtkm::TypeListTagScalar>);
  typedef void ExecutionSignature(_1, _2, _3, _4);

  template<typename T1, typename T2, typename T3, typename T4>
  VTKM_EXEC_EXPORT
  void operator()(T1 real, T2 imaginary,
                  T3 &magnitude, T4 &phase) const
```

1
2
3.1
3.2
3.3
3.4

Worklet Arguments

- Argument type based on concept defined in `ControlSignature`
- A field in the same domain (e.g. `FieldIn` in a `WorkletMapField` or `FieldInTo` in a `WorkletMapTopology`)
  - Base type in the corresponding array (e.g. `vtkm::Float32`)
- A field associated with the "from" topology (e.g. `FieldInFrom` in a `WorkletMapTopology`)
  - An object that has overloaded `operator[]` to behave like array or `vtkm::Vec`.
  - Also has `ComponentType` typedef and `GetNumberOfComponents` method.

Working with Cells

Cell Shapes

- VTK-m cell shapes copy those of VTK
- Basic shapes defined in vtkm/CellShape.h
- Every cell shape has an enum identifier
  - e.g. `vtkm::CELL_SHAPE_TRIANGLE`, `vtkm::CELL_SHAPE_HEXAHEDRON`
- Every cell shape has a tag struct
  - e.g. `vtkm::CellShapeTagTriangle`, `vtkm::CellShapeTagHexahedron`
- All cell shape tags have a member Id set to the identifier
  - `vtkm::CellShapeTagTriangle::Id == vtkm::CELL_SHAPE_TRIANGLE`
- For a constant cell shape identifier, can get tag with `vtkm::CellShapeIdToTag`
  - `vtkm::CellShapeIdToTag<CELL_SHAPE_TRIANGLE>::Tag` is typedef'ed to `vtkm::CellShapeTagTriangle`

Generic Cell Shape

- `vtkm::CellShapeTagGeneric` is a special shape tag
  - Has no corresponding integer identifier
  - `Id` field set at runtime
- Can write specialized cell functions on generic shapes using `vtkmGenericCellShapeMacro`

Cell Traits

- Defined in vtkm/CellTraits.h
- `vtkm::CellTraits<>` template provides static cell information
  - `TOPOLOGICAL_DIMENSIONS`: 3 for polyhedra, 2 for polygons, 1 for lines, 0 for vertices
  - `TopologicalDimensionsTag`: Set to `vtkm::CellTopologicalDimensionsTag<TOPOLOGICAL_DIMENSIONS>`
  - `IsSizedFixed`: Set to `vtkm::CellTraitsTagSizeFixed` if there is a static number of points, `vtkm::CellTraitsTagSizeVariable` otherwise
  - `NUM_POINTS`: Set to the number of points in the cell. Only defined if the size is fixed

Using Cell Shapes in Worklets

- Use the `ExecutionSignature` tag `CellShape`
  - Defined in worklet types that support it (e.g. `WorkletMapTopology`)

```
struct MyWorklet : public vtkm::worklet::WorkletMapTopology<vtkm::TopologyElementTagPoint,
                                                            vtkm::TopologyElementTagCell>
{
  typedef void ControlSignature(TopologyIn topology,
                                FieldInFrom<Scalar> inField,
                                FieldOut<Scalar> outCells)
  typedef _3 ExecutionSignature(CellShape, _2);

  template<typename CellShapeTag, typename InValues>
  VTKM_EXEC_EXPORT
  T operator()(CellShapeTag shape, const InValues &inValues) const
  {
    // Operate using shape...
```

Cell Operations

- `#include <vtkm/exec/ParametricCoordinates.h>`
  - Convert between world coordinates and parametric coordinates (locations in the cell are always in the range [0,1])
- `#include <vtkm/exec/CellInterpolate.h>`
  - Given a group of field coordinates and a parametric coordinate, interpolates the field to that point.
- `#include <vtkm/exec/CellDerivative.h>`
  - Given a group of field coordinates and a parametric coordinate, computes the derivative (gradient) of the field at that point.

Error Handling

Errors Signaled with Exceptions

- On error, Dax toolkit throws a subclass of `vtkm::cont::Error`

```
#include <vtkm/cont/Error.h>

int main(int argc, char **argv)
{
  try
    {
    // Do something cool with VTK-m
    // ...
    }
  catch (vtkm::cont::Error error)
    {
    std::cout << error.GetMessage() << std::endl;
    return 1;
    }
  return 0;
}
```

Types of Errors

- `vtkm::cont::ErrorControlAssert`
  - VTK-m fails an assertion. This might be a bad parameter value (such as an index out of range) with the check removed in release builds.
- `vtkm::cont::ErrorControlBadType`
  - A VTK-m encounters an unexpected, invalid, or unknown type.
- `vtkm::cont::ErrorControlBadValue`
  - A VTK-m function or method encounters an invalid value.
- `vtkm::cont::ErrorControlInternal`
  - VTK-m detects an internal state that should never be reached. This error usually indicates a bug in VTK-m or, at best, VTK-m failed to detect an invalid input.
- `vtkm::cont::ErrorControlOutOfMemory`
  - A VTK-m function or method tries to allocate an array and fails.

Reporting Errors in Worklets

- Exceptions cannot be thrown in the execution environment
  - Not supported in CUDA. Problematic with multiple threads.
- All worklets have a method named `RaiseError`
  - Call this method with a message string.
- In the control environment, a `vtkm::cont::ErrorExecution` will be thrown with the given message
- Behaves as if the error was thrown in the worklet
  - Be aware, raising an error might not actually halt any execution.

```
VTKM_EXEC_EXPORT
T operator()(T x) const
{
  if (x < 0)
  {
    this->RaiseError("Cannot take square root of negative number.");
  }
  return vtkm::math::Sqrt(x);
}
```

Asserts

- VTKM_ASSERT_CONT defined in vtkm/cont/Assert.h
  - Behaves like POSIX assert except that it throws `vtkm::cont::ErrorControlAssert` instead of exiting the program.
  - Will be removed when NDEBUG is defined.
    - CMake adds this to Release builds.
- VTKM_ASSERT_EXEC defined in vtkm/exec/Assert.h
  - Takes a second argument that is a worklet.
  - Behaves like POSIX assert except that it throws `VTKM::cont::ErrorExecution` instead of exiting the program.
  - Will be removed when NDEBUG is defined.
    - CMake adds this to Release builds.

Timers

Timing Parallel Code

- `vtkm::cont::Timer` safely times parallel VTK-m algorithms
  - Starts on construction or `Reset`. Reports on `GetElapsedTime`.

```
vtkm::cont::Timer<> timer;
dispatcher.Invoke(
    grid.GetCoordinateSystem().GetData()GetPointCoordinates(),
    results);
// This call makes sure data is pulled back to the host
// in a host/device architecture.
results.GetPortalConstControl();
vtkm::Float32 elapsedTime = timer.GetElapsedTime();
```