

VTK-M: A FOUNDATION

Kenneth Moreland, et al.

SDAV Vis Teleconference
Tuesday 15, 2014

VTK-m Project

Target Multi and Many core architectures.

Combines the strengths of multiple projects:

EAVL, DAX, and PISTON

Reduce the complexity of writing highly concurrent code.

Funding

VTK-m development made part of recent XVis proposal to ASCR Scientific Data Management, Analysis and Visualization at Extreme Scale 2 call

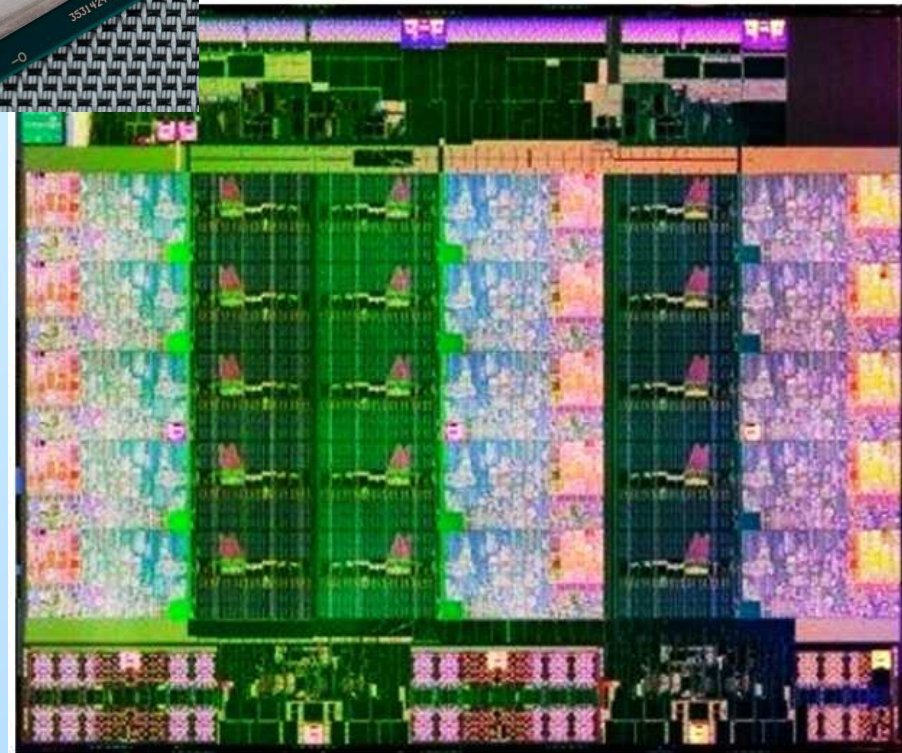
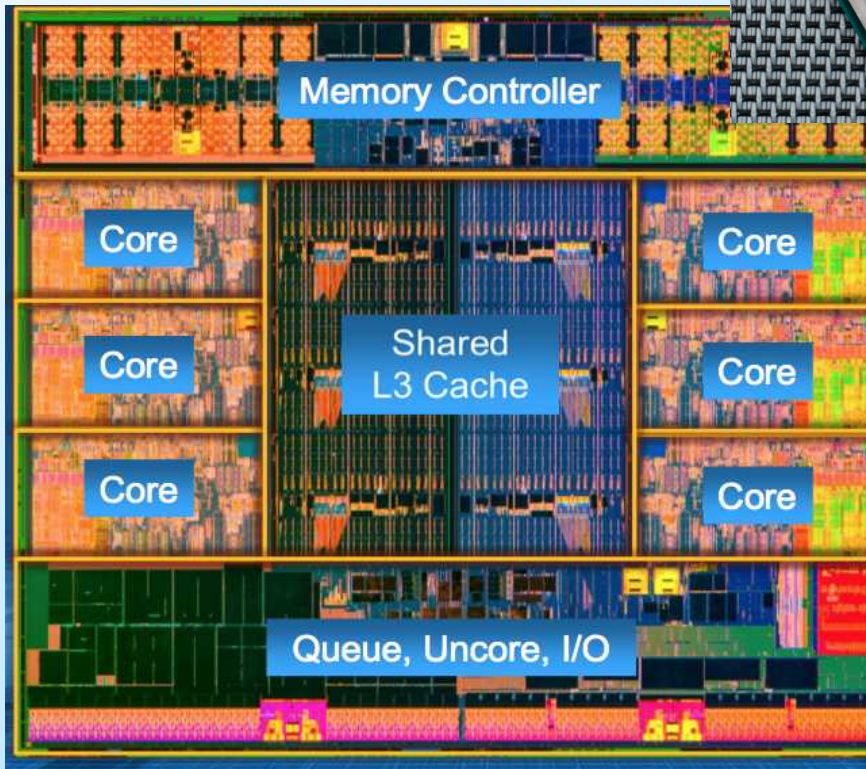
Recommended for award

Funding expected to start in September

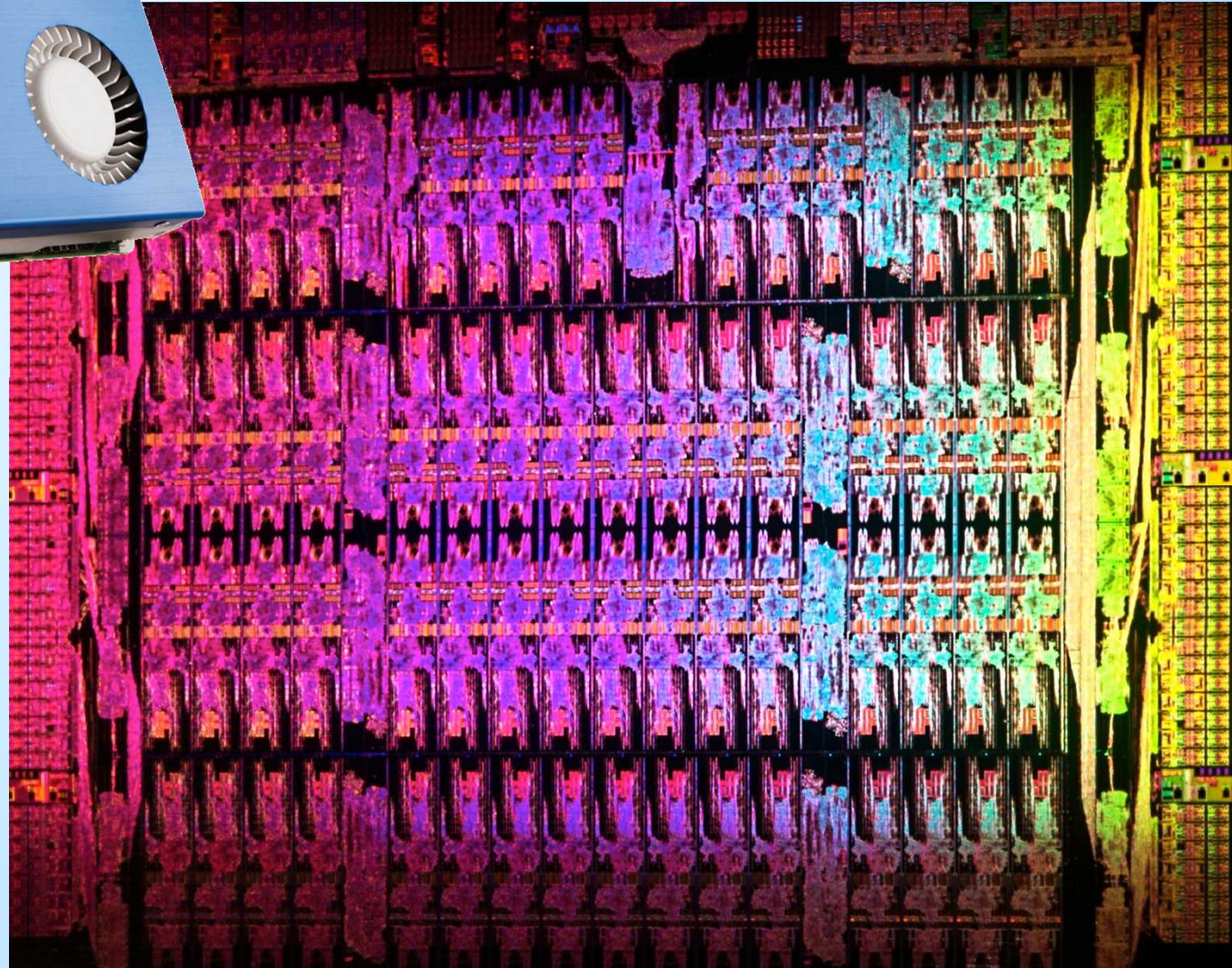
Multi and Many Core

What exactly are we talking about

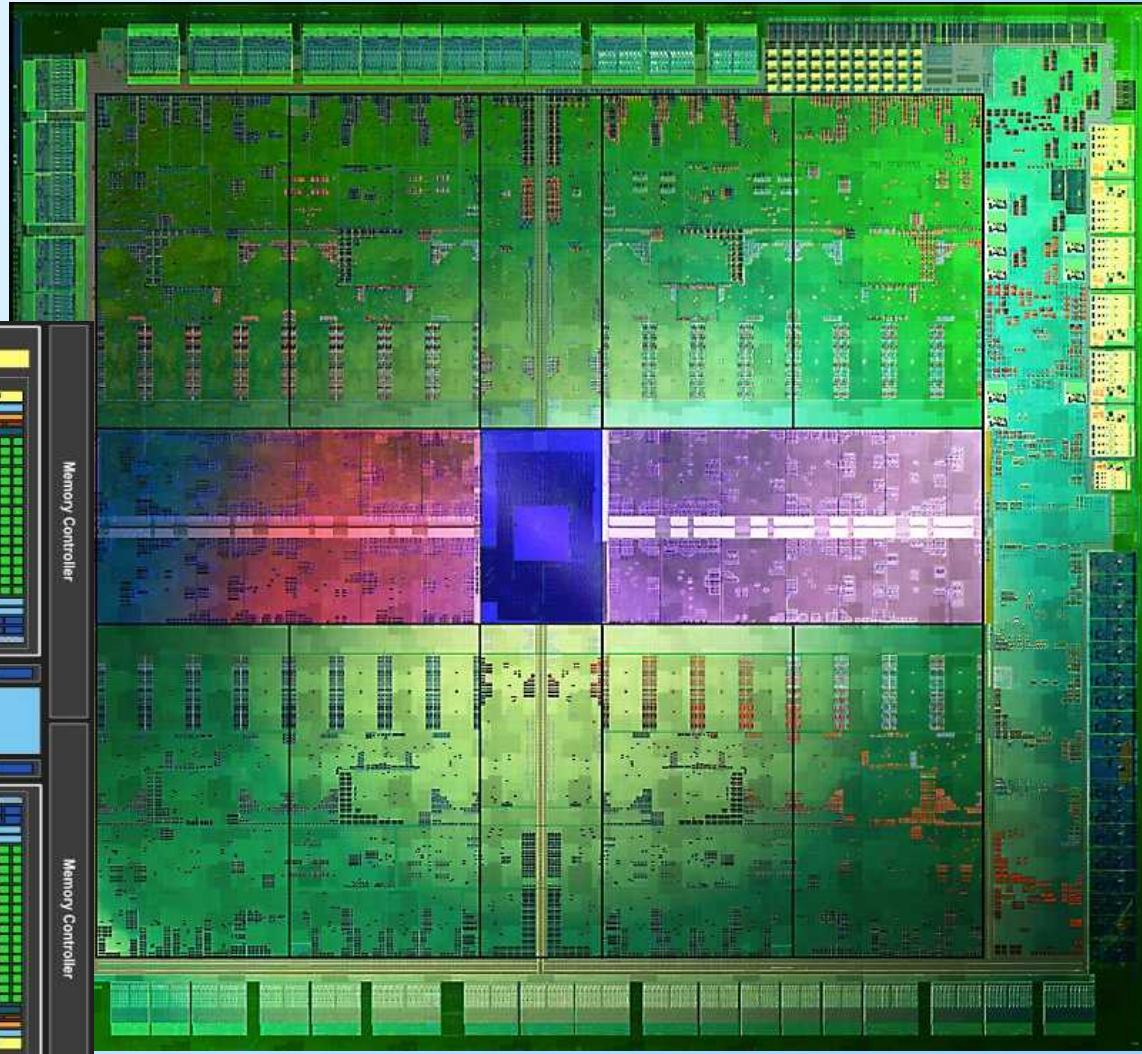
Multi Core



Multi Core

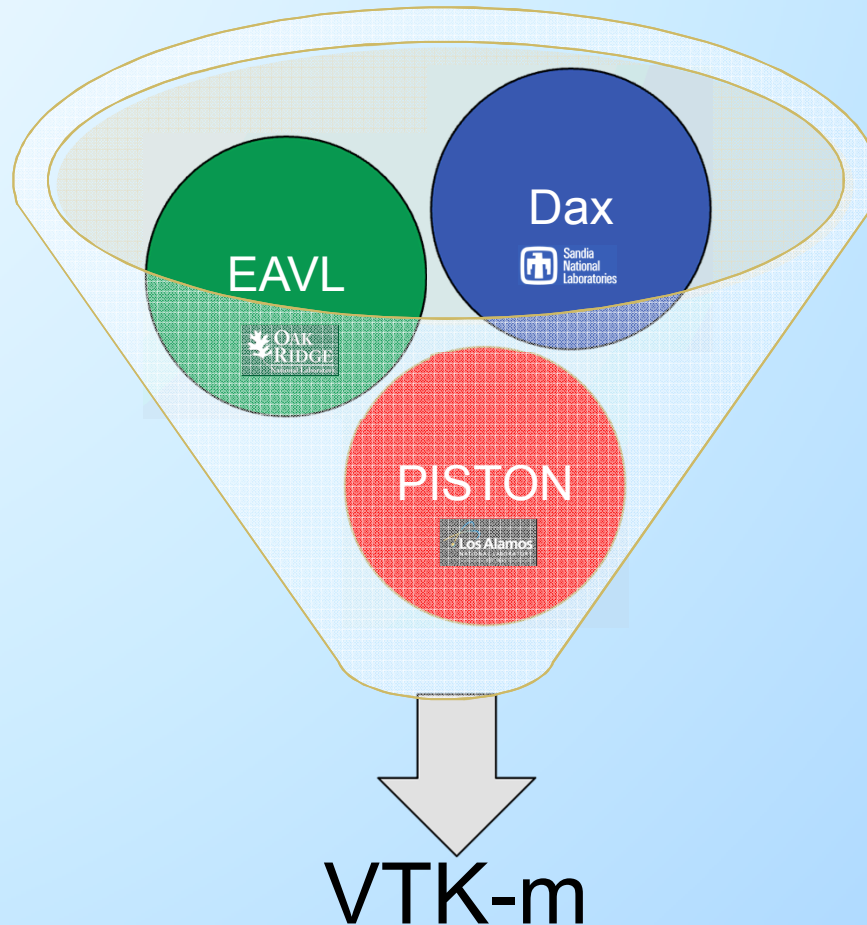


Many Core

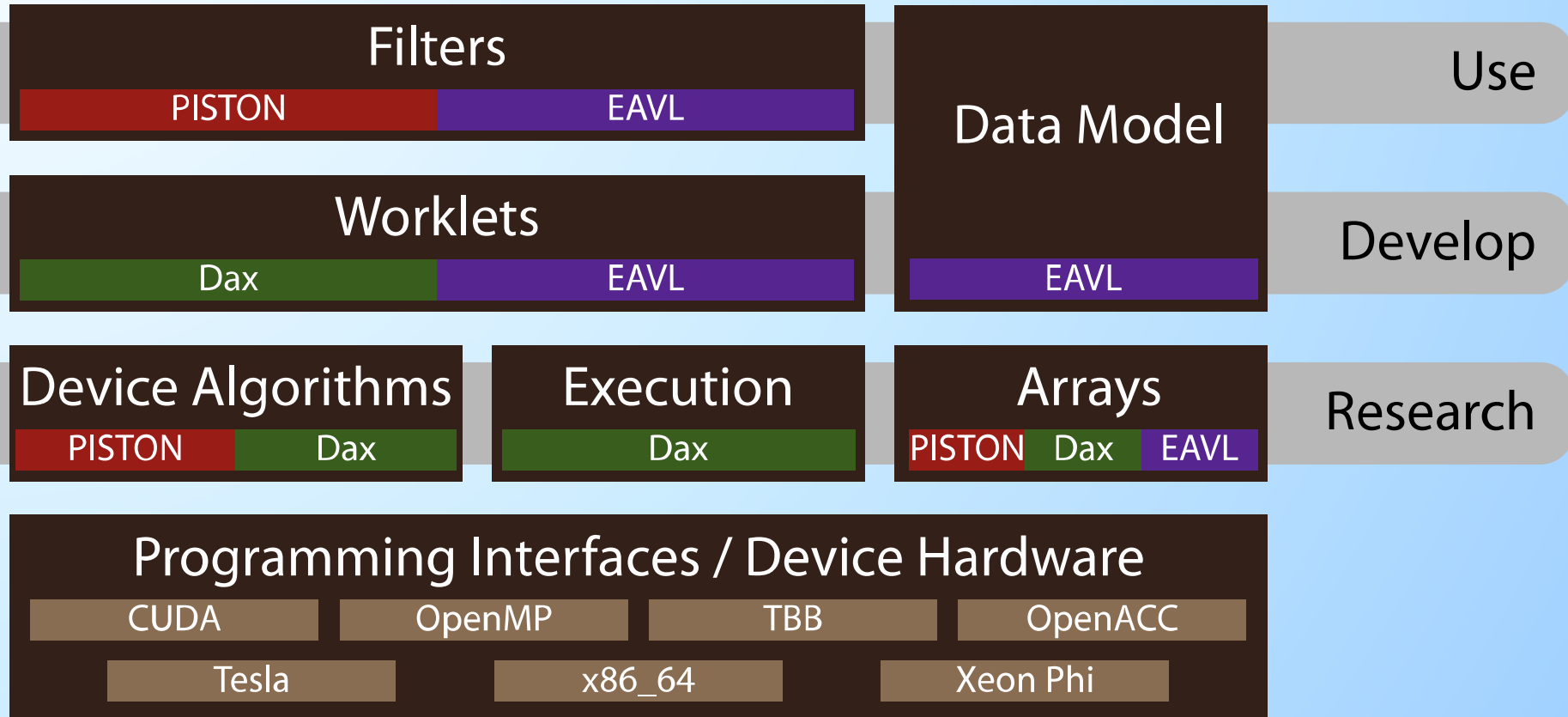


VTK-m: A Foundation

Combines the strengths of multiple projects



VTK-m Architecture



Highly concurrent code

High Level Concept

Decompose algorithms into sections that can run small section of data.

The approach is essentially that same as presented by Baker and colleagues, functional mapping [Baker, et al. 2010]



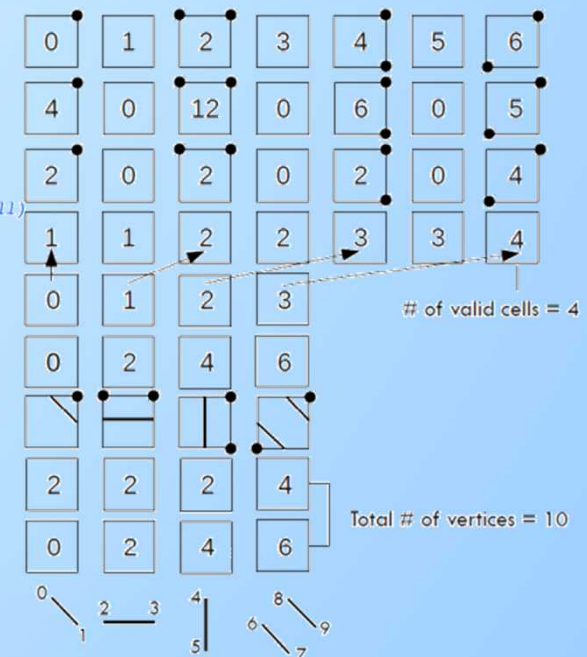
functor()

Encapsulate Parallel Operations

Visualization algorithms can be composed of base operations like scan, sort, for each, reduce, etc.

Based on work of Belloch [1990]

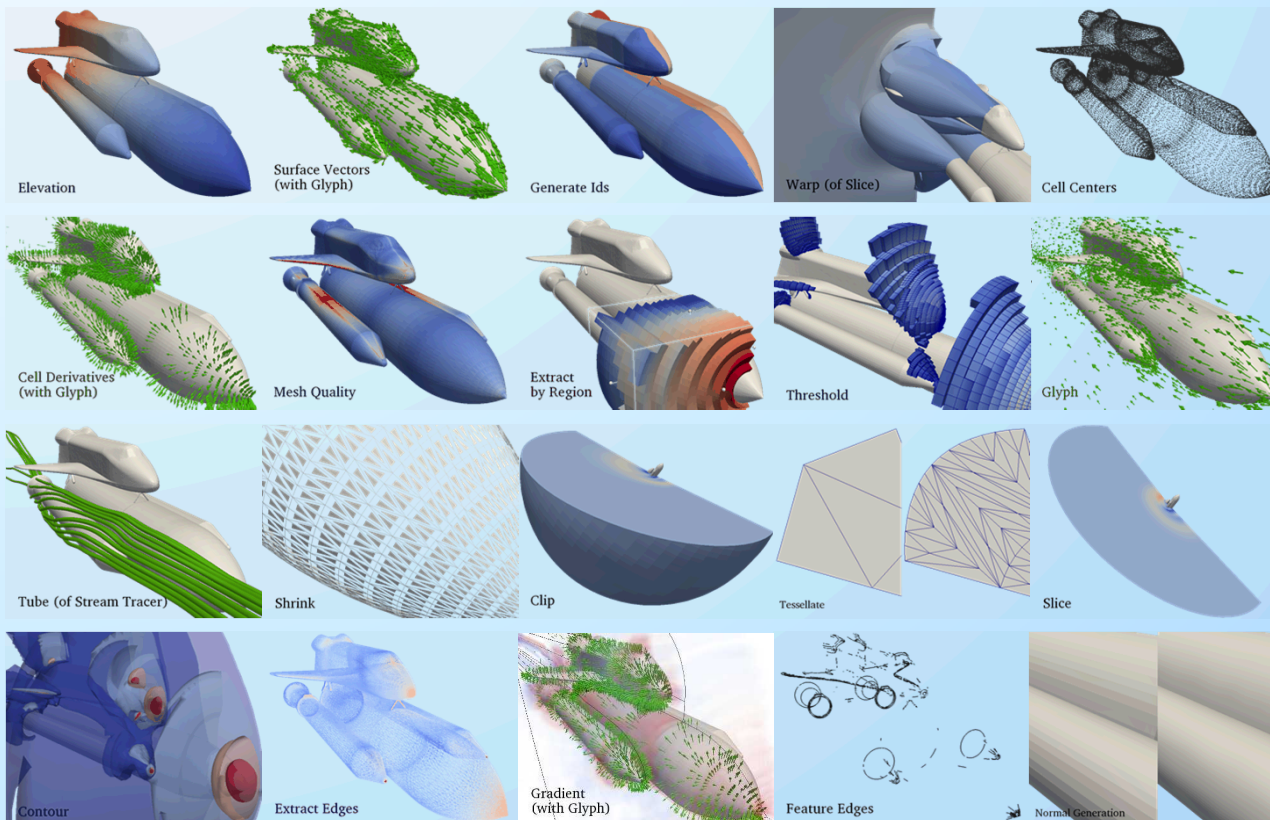
1. input
transform(classify_cell)
2. caseNums
3. numVertices
transform_inclusive_scan(is_valid_cell)
4. validCellEnum
5. CountingIterator
upper_bound
6. validCellIndices
7. numVerticesCompacted
make_permutation_iterator
exclusive_scan
8. numVerticesEnum
for_each(isosurface_function)
9. outputVertices



Lo, Swell, Ahrens. "PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operations," 2012

Simplifying Algorithm Development

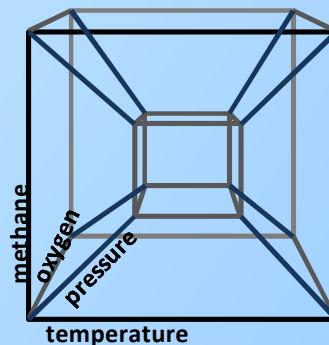
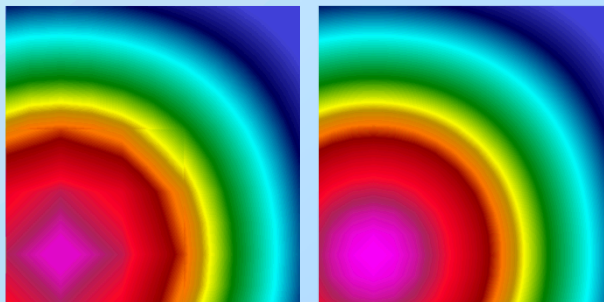
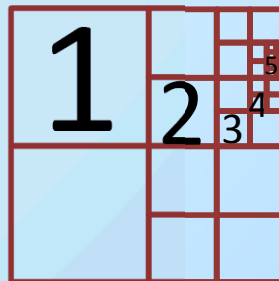
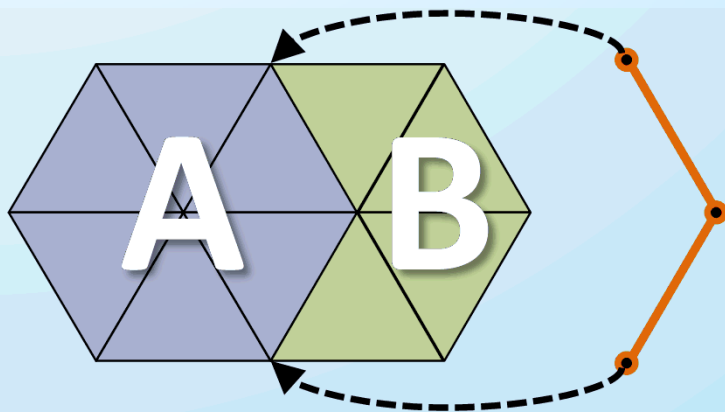
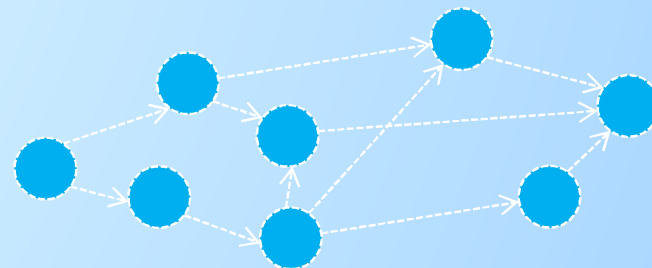
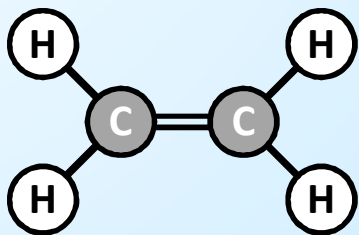
Many visualization algorithms share algorithm structure. These patterns can be directly implemented in the toolkit.



Moreland, Geveci, Ma, Maynard. "A Classification of Scientific Visualization Algorithms for Massive Threading," 2013

Powerful, Flexible Data Structures

Need to represent many structure types but with consistent access that works on all devices



VTK-m Code Repository

<http://public.kitware.com/gitweb?p=vtkm.git>

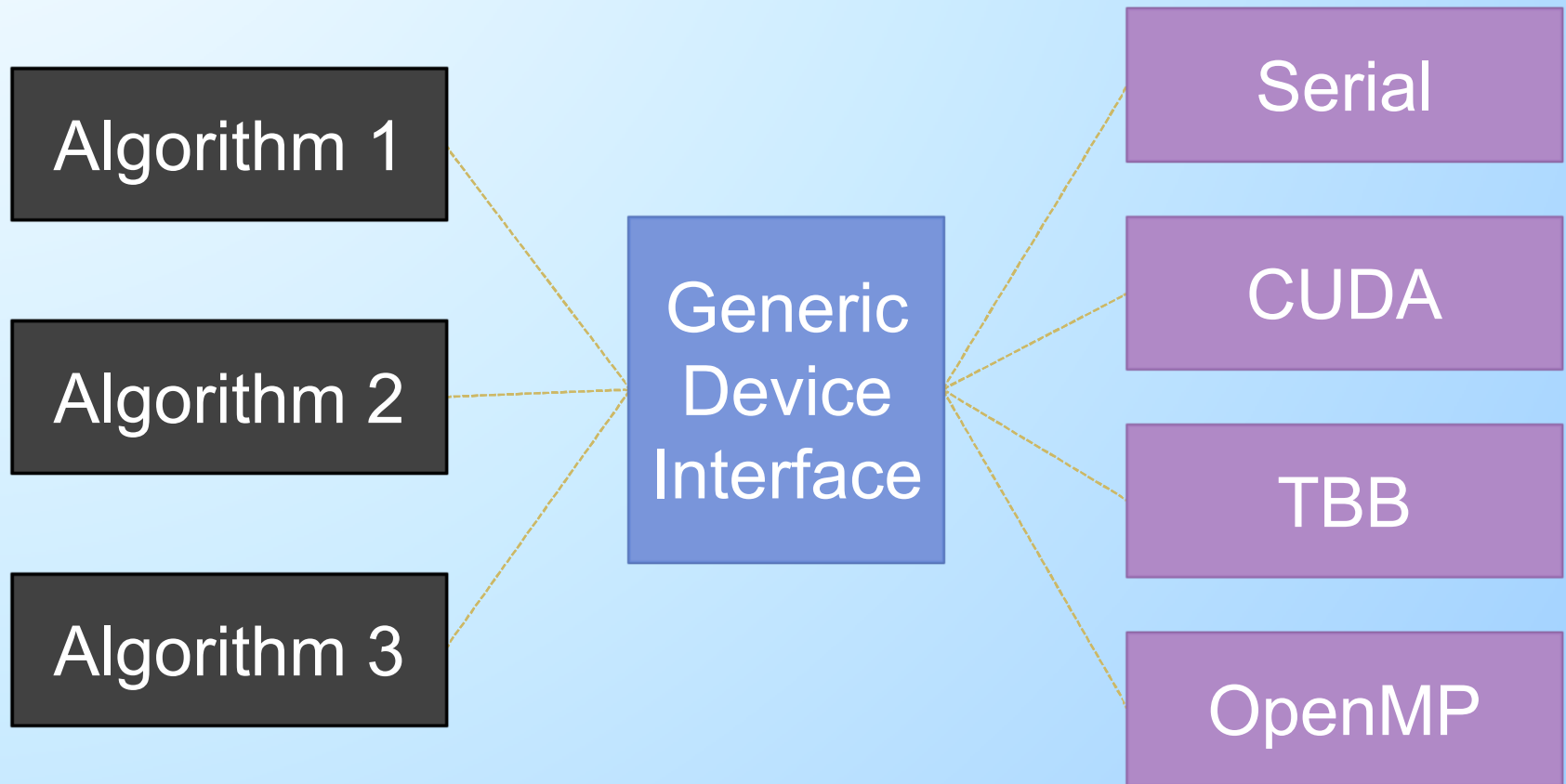
Currently contains basic foundations to combine features from Dax, PISTON, EAVL

Expect to hit ground running when XVis starts

The screenshot shows the VTK-m Git repository page on public.kitware.com. The page title is "projects / vtkm.git / summary". It includes a search bar and a "commit" button. The "description" section lists the owner as Kitware, Inc. and the last change as "Thu, 10 Jul 2014 16:58:04 +0000 (10:58 -0600)". The "shortlog" section displays a list of recent commits, including "Merge branch 'icc-fix'", "Make floating point comparison more tolerant in UnitTes...", "Merge branch 'expand-function-interface'", "Merge branch 'cdash_support'", "Add the correct copyright statement to CTestConfig.", "Merge branch 'cdash_support'", "Add a CDash support to vtkm.", "If pyexpander is available, run it and check the source...", "Replace Boost preprocessor iteration with macro expansi...", "Merge branch 'container-to-storage'", "Change ArrayContainerControl to Storage.", "Merge branch 'windows_function_interface'", "MSVC 2010 has troubles with < operators in the middle...", "Merge branch 'count-vectors'", "Fix a test object to handle changes in the counting...", and "Minor change that allows vector types to be used in...". The "heads" section lists the current branches: "master", "function_ptr_issue", and "dynamic-arrays".

Interchangeable Devices

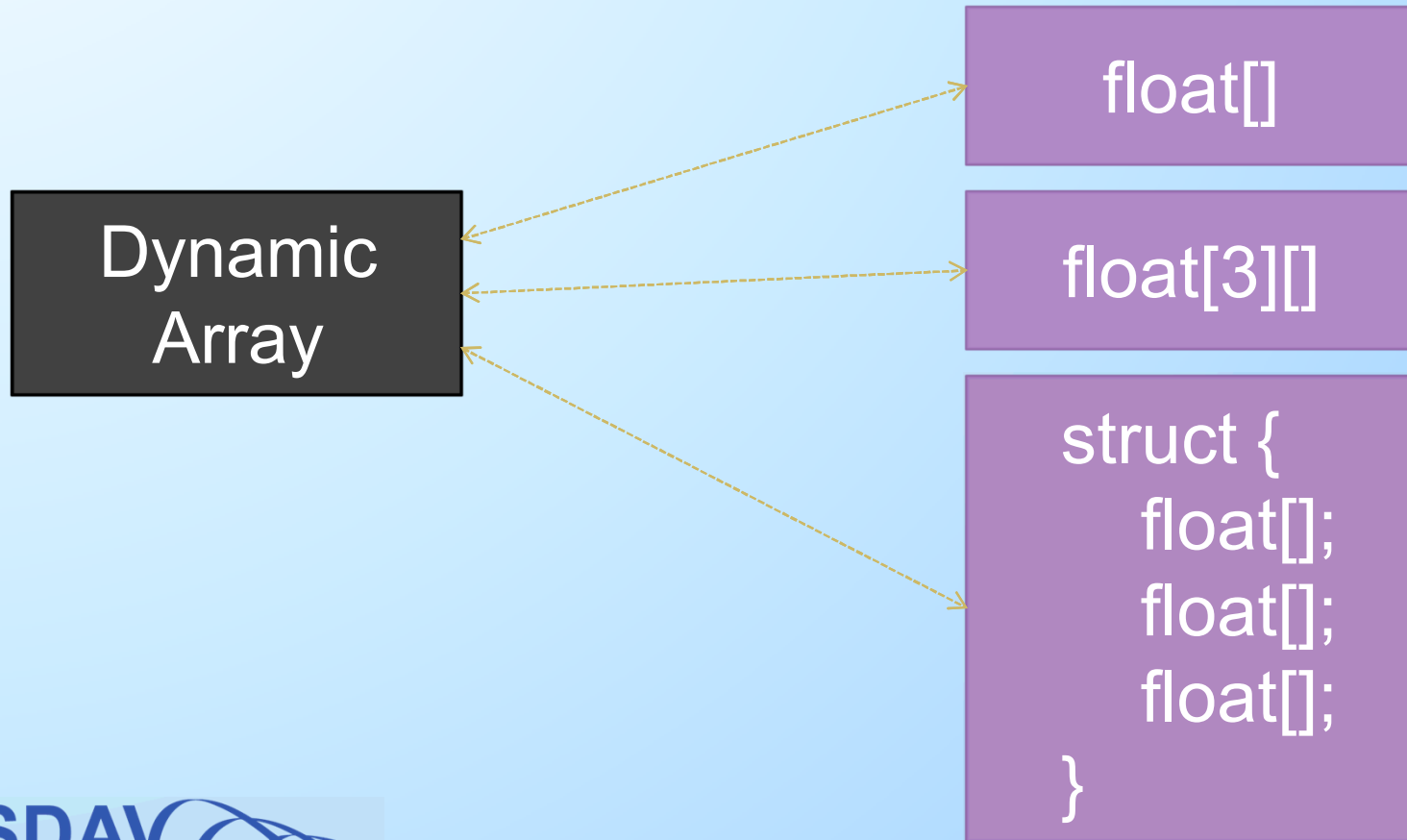
Cannot rewrite every algorithm for every device
Use an abstract device to minimize porting



Enable Polymorphic Types

Efficient code likes templates for direct data access (especially on GPUs)

Data types are often unknown until run time



Documentation

Writing documentation
for VTK-m as we go

Important for
communication

All existing code well
documented with
examples (currently
about 60 pages)

2.4.2 Vector Traits

The `vtkm::VectorTraits<T>` templated class provides information and accessors to vector and tuple types. It contains the following elements.

ComponentType This type is set to the type for each component in the vector. For example, a `vtkm::Vector3` has `ComponentType` defined as `vtkm::Scalar`.

NUM_COMPONENTS An integer specifying how many components are contained in the vector.

HasMultipleComponents This type is set to either `vtkm::VectorTraitsTagSingleComponent` if the vector length is size 1 or `vtkm::VectorTraitsTagMultipleComponents` otherwise. This tag can be useful for creating specialized functions when a vector is really just a scalar.

GetComponent A static method that takes a vector and returns a particular component.

SetComponent A static method that takes a vector and sets a particular component to a given value.

ToTuple A static method that converts a vector of the given type to a `vtkm::Tuple`.

The definition of `vtkm::VectorTraits` for `vtkm::Id3` could like something like this.

Example 2.10: Definition of `vtkm::VectorTraits<vtkm::Id3>`.

```
namespace vtkm {  
  
template<>  
struct VectorTraits<vtkm::Id3>  
{  
    typedef vtkm::Id ComponentType;  
    static const int NUM_COMPONENTS = 3;  
    typedef VectorTraitsTagMultipleComponents HasMultipleComponents;  
  
    VTKM_EXEC_CONT_EXPORT  
    static vtkm::Id &GetComponent(vtkm::Id3 &vector, int component) {  
        return vector[component];  
    }  
  
    VTKM_EXEC_CONT_EXPORT  
    static void SetComponent(vtkm::Id3 &vector, int component, vtkm::Id value) {  
        vector[component] = value;  
    }  
  
    VTKM_EXEC_CONT_EXPORT  
    static vtkm::Tuple<vtkm::Id,3> ToTuple(const vtkm::Id3 &vector) {  
        return vector;  
    }  
};  
  
} // namespace vtkm
```

The real power of vector traits is that they simplify creating generic operations on any type that can look like a vector. This includes operations on scalar values as if they were