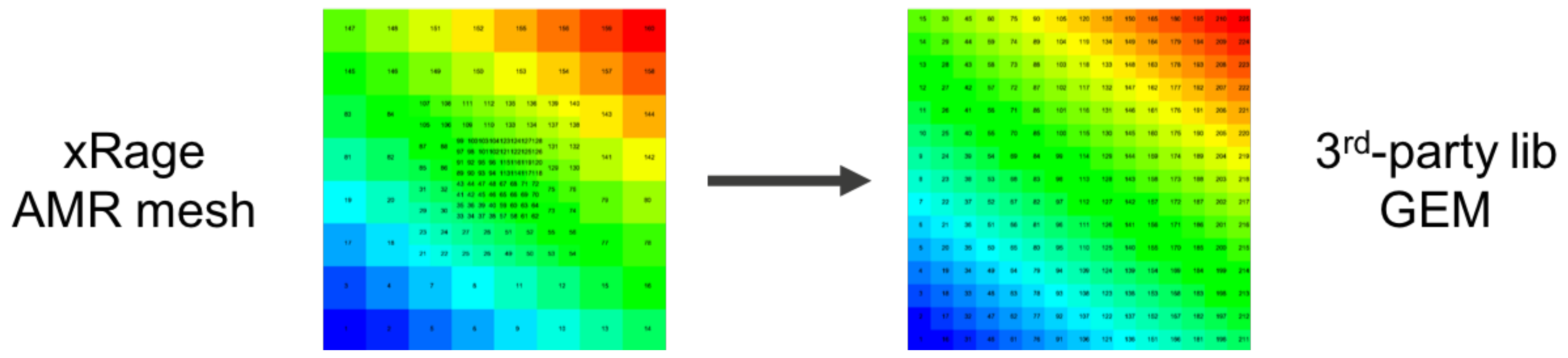
Linking a Next-Gen Remap Library into a Long-Lived Production Code

Charles R. Ferenbaugh and Brendan K. Krueger
Los Alamos National Laboratory

Supercomputing 2019
November 19, 2019

Our task: Add Portage remapping to xRage

- The xRage application runs on an AMR mesh
- Third-party libraries often use their own mesh, such as a Generalized Eulerian Mesh (GEM)
- xRage must map fields between the two meshes

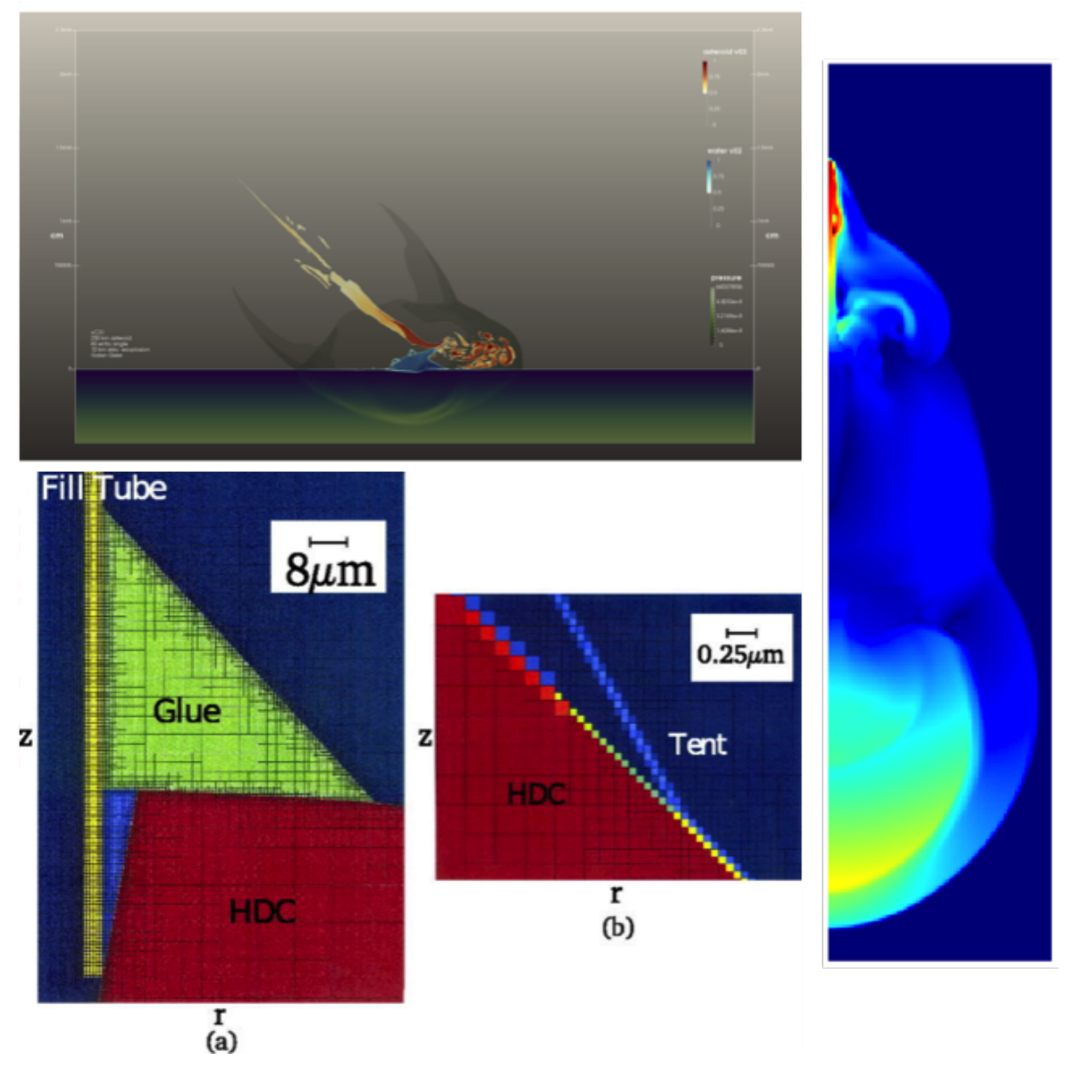


- The current xRage mapper was implemented in a short timeframe
  - Not well understood by current code team
  - Not easily maintainable, extensible
- Using the new Portage library would be much more flexible

How can we take two codes that are so different, and make them work together?

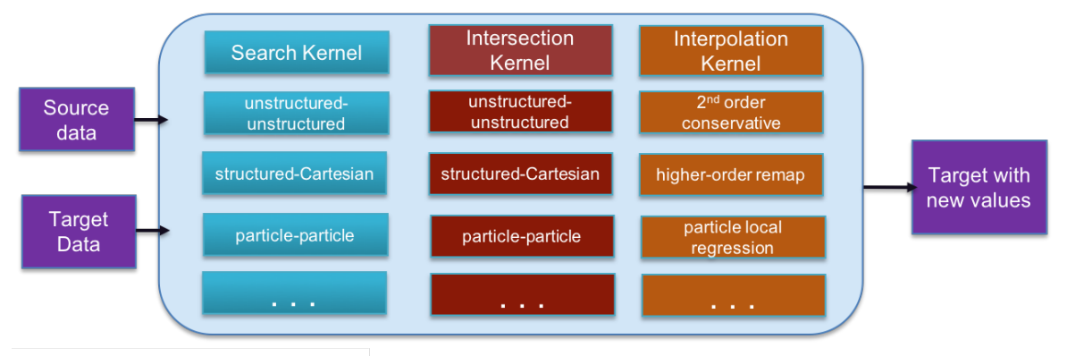
Background: The xRage application

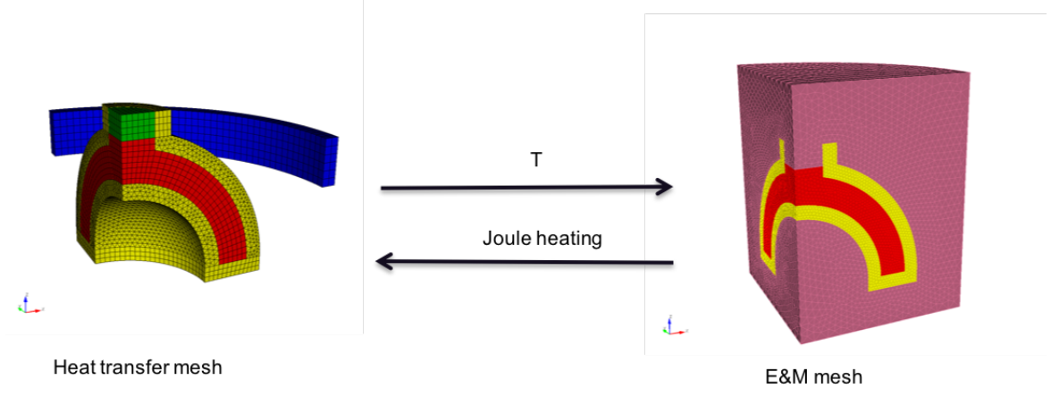
- xRage is an Eulerian AMR radiation-hydrodynamics code
- Original code written ~1990
- Contains about 470K SLOC
  - Not counting numerous third-party libraries
- Mostly Fortran 90, some C/C++
- MPI-only parallelism
- Code modernization is ongoing



Background: The Portage remapping library

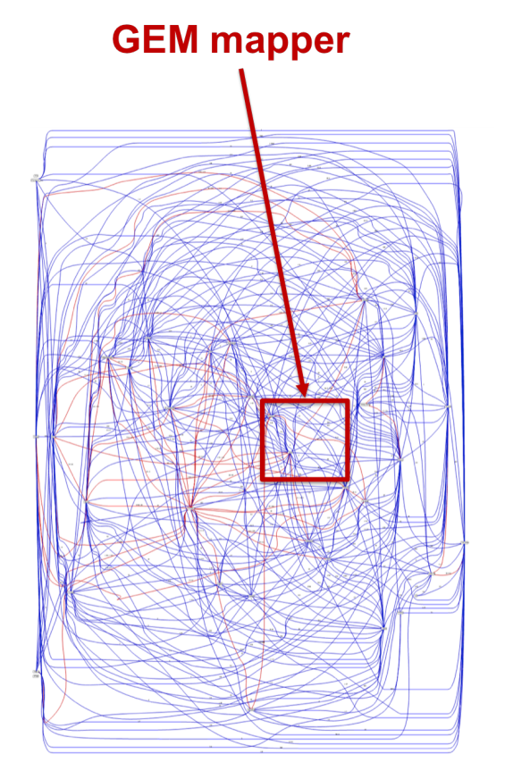
- Portage is a modern framework for remapping and linking
- Development started in 2015
- Modular, extensible design
- Contains about 14K SLOC
- Written in C++, makes heavy use of classes and templates
- MPI+OpenMP parallelism

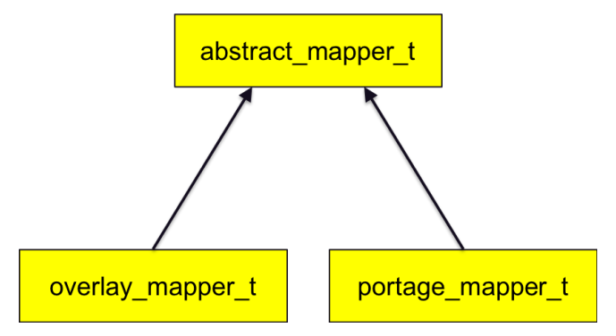




Legacy mapper cleanup

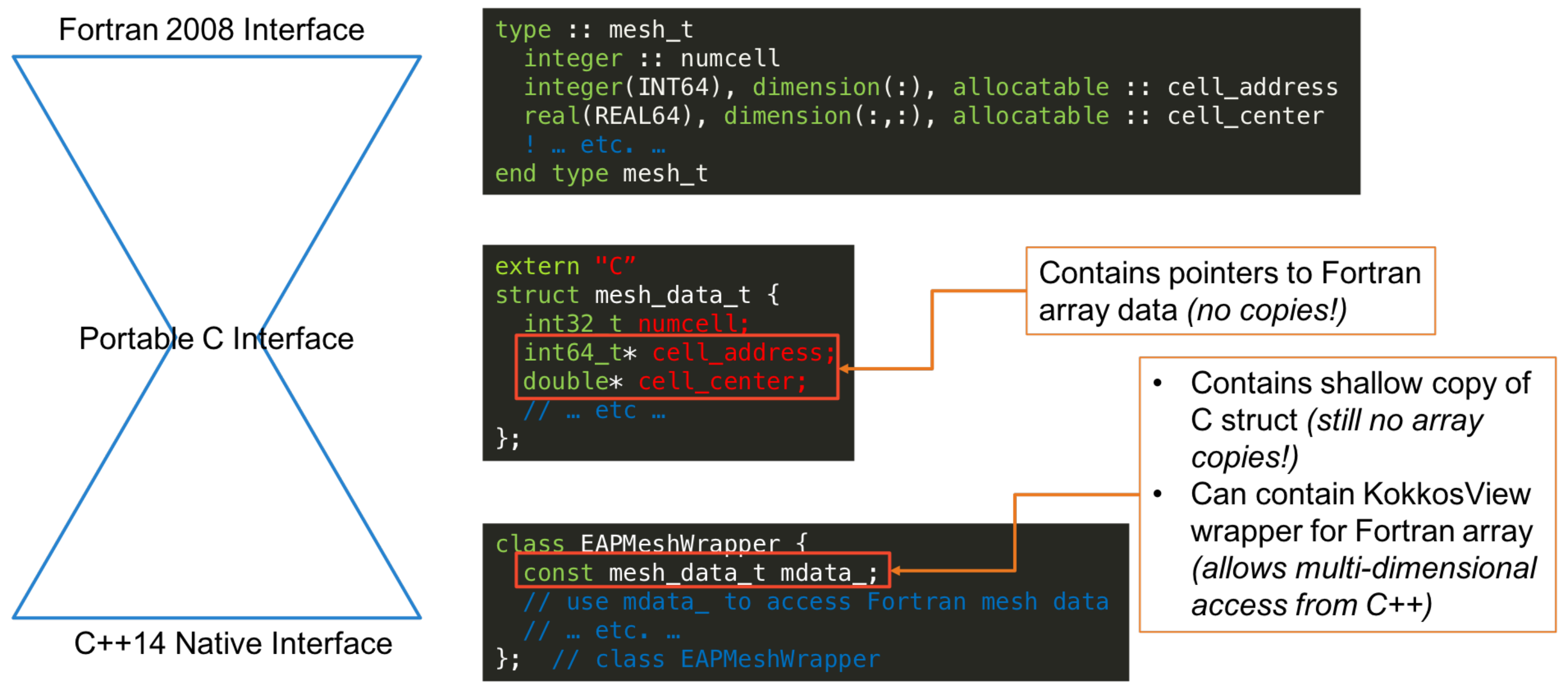
- The legacy GEM mapper had several design limitations:
  - Not encapsulated from the rest of xRage
  - Used private module data for everything
  - Not unit-testable
- We did major refactoring to address these issues
- We also created a remapper base class, allowing us to switch easily between legacy and Portage mappers





Creating Fortran/C++ interfaces

- We wrote C++ wrapper interfaces to native Fortran data
  - Uses C interop features from Fortran 2003, and the "Hourglass Interface" design pattern
  - Avoids making copies of large arrays
  - Uses KokkosViews for multidimensional arrays in C++



```
type :: mesh_t
  integer :: numcell
  integer(INT64), dimension(:), allocatable :: cell_address
  real(REAL64), dimension(:,:), allocatable :: cell_center
  ! … etc. …
end type mesh_t
```

```
extern "C"
struct mesh_data_t {
  int32_t numcell;
  int64_t* cell_address;
  double* cell_center;
  // … etc …
};
```

Contains pointers to Fortran array data *(no copies!)*

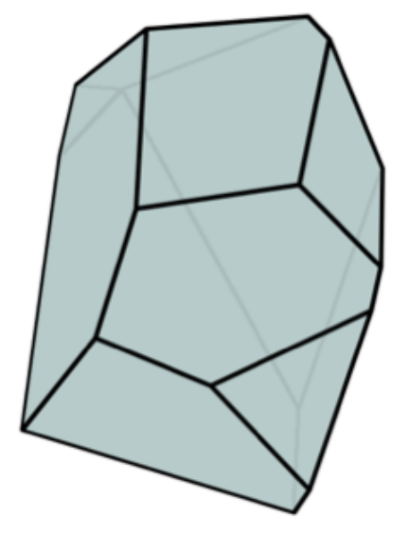
```
class EAPMeshWrapper {
  const mesh_data_t mdata_;
  // use mdata_ to access Fortran mesh data
  // … etc. …
};  // class EAPMeshWrapper
```

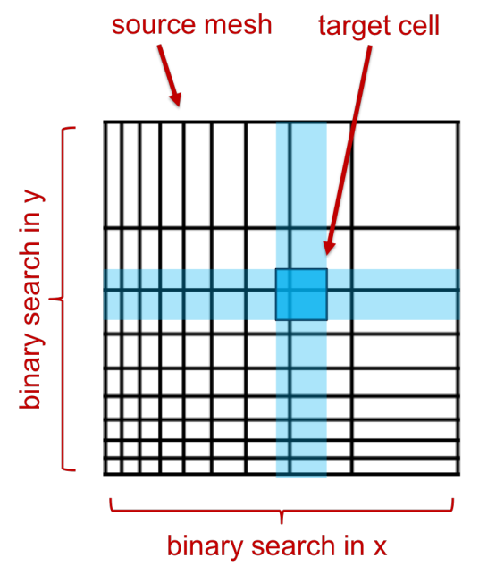
- Contains shallow copy of C struct *(still no array copies!)*
- Can contain KokkosView wrapper for Fortran array *(allows multi-dimensional access from C++)*

- We also wrote similar interface wrappers for subroutines

xRage extensions to Portage

- xRage needed features that Portage didn't (yet) support:
  - Support for cylindrical (r-z) geometry
  - Specialized intersector for boxes, in place of general polygon/polyhedron intersector
  - Specialized search/distribute for GEM meshes, in place of general kD-tree search
- We developed these as extensions in xRage
- We're migrating all of them back to Portage
  - Taking advantage of Portage extensible design





Current status

- Code is working, passing tests in xRage unit test framework
- Supports 2D and 3D remaps (1D in progress)
- Supports MPI parallelism
- Integration into physics packages is in progress

Initial timing results: Portage vs. legacy mapper

Case 1
AMR: 2.8M cells, distributed
GEM: 200K cells, distributed
MPI ranks: 800

Case 2
AMR: 2.9M cells, distributed
GEM: 200K cells, single rank
MPI ranks: 576

	Average time per call (ms)					
	Case 1			Case 2		
map direction	legacy	Portage	speedup	legacy	Portage	speedup
AMR to GEM	38.5	18.4	2.09x	56.4	21.5	2.62x
GEM to AMR	30.8	1.8	17.01x	6754.5	302.2	22.35x

Memory usage (case 2): legacy ≈ 1.83 Gb, Portage ≈ 0.24 Gb

Summary

- xRage-Portage remap improves performance and memory use over legacy mapper
- Fortran/C++ interface strategy is effective
- Portage modular design allowed us to add xRage-specific customizations easily
- xRage can now leverage current and future Portage features
  - Including support for many-core (current) and GPU (future)

This work will provide a model for more next-generation packages to be integrated into production codes

Acknowledgements

Thanks to:

- Andrew Gaspar and Robert Pavel, for working through Fortran/C++ interface issues elsewhere in xRage
- Neil Carlson, for writing a Portage-Truchas link that provided some ideas for this work
- The xRage and Portage code teams

Los Alamos National Laboratory

LA-UR-19-27263