Hewlett Packard Enterprise

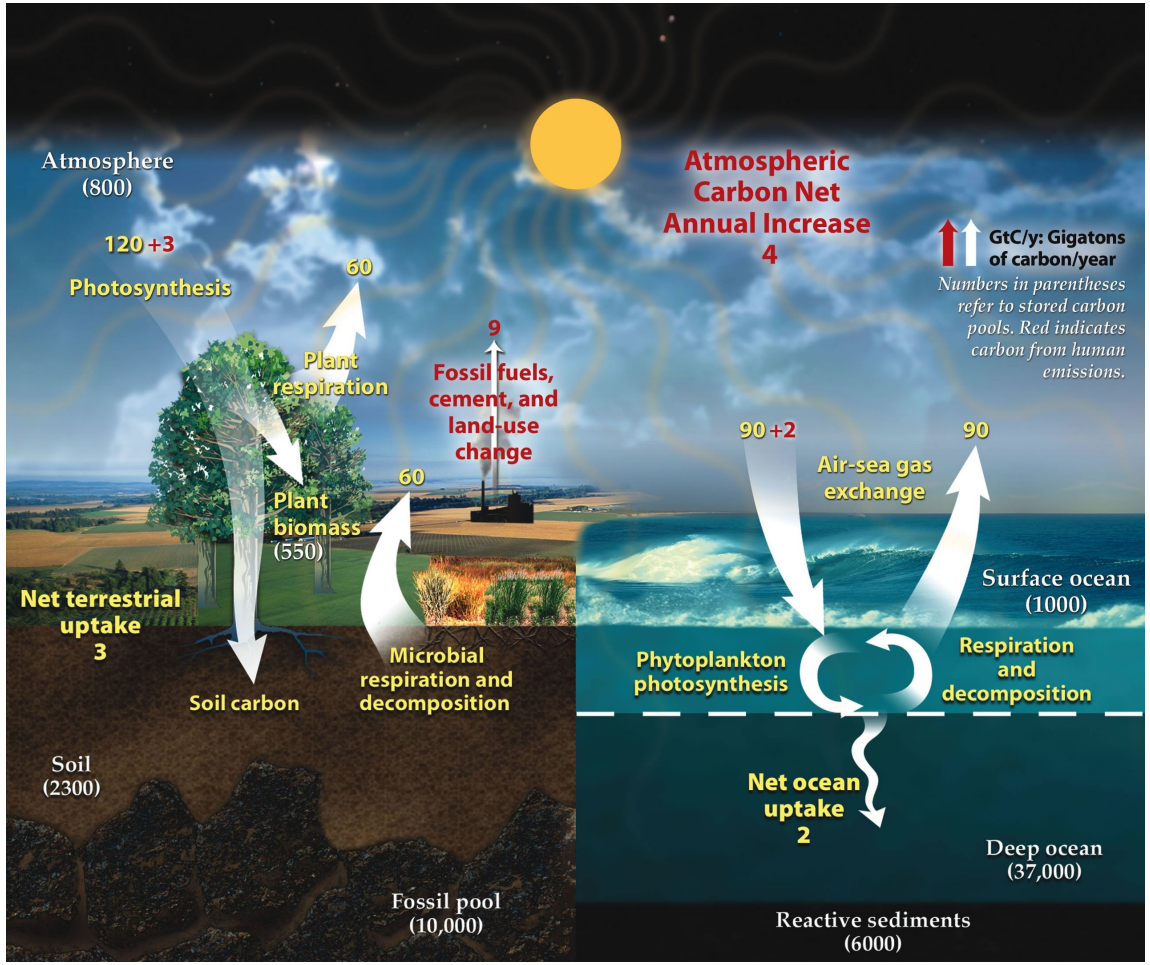
MARBLChapel: Fortran-Chapel Interoperability in an Ocean Simulation

Brandon Neth, HPE; Scott Bachman, [C]Worthy; Michelle Mills Strout, HPE

Big Picture: Modeling Climate Interventions like Carbon Dioxide Removal (CDR)

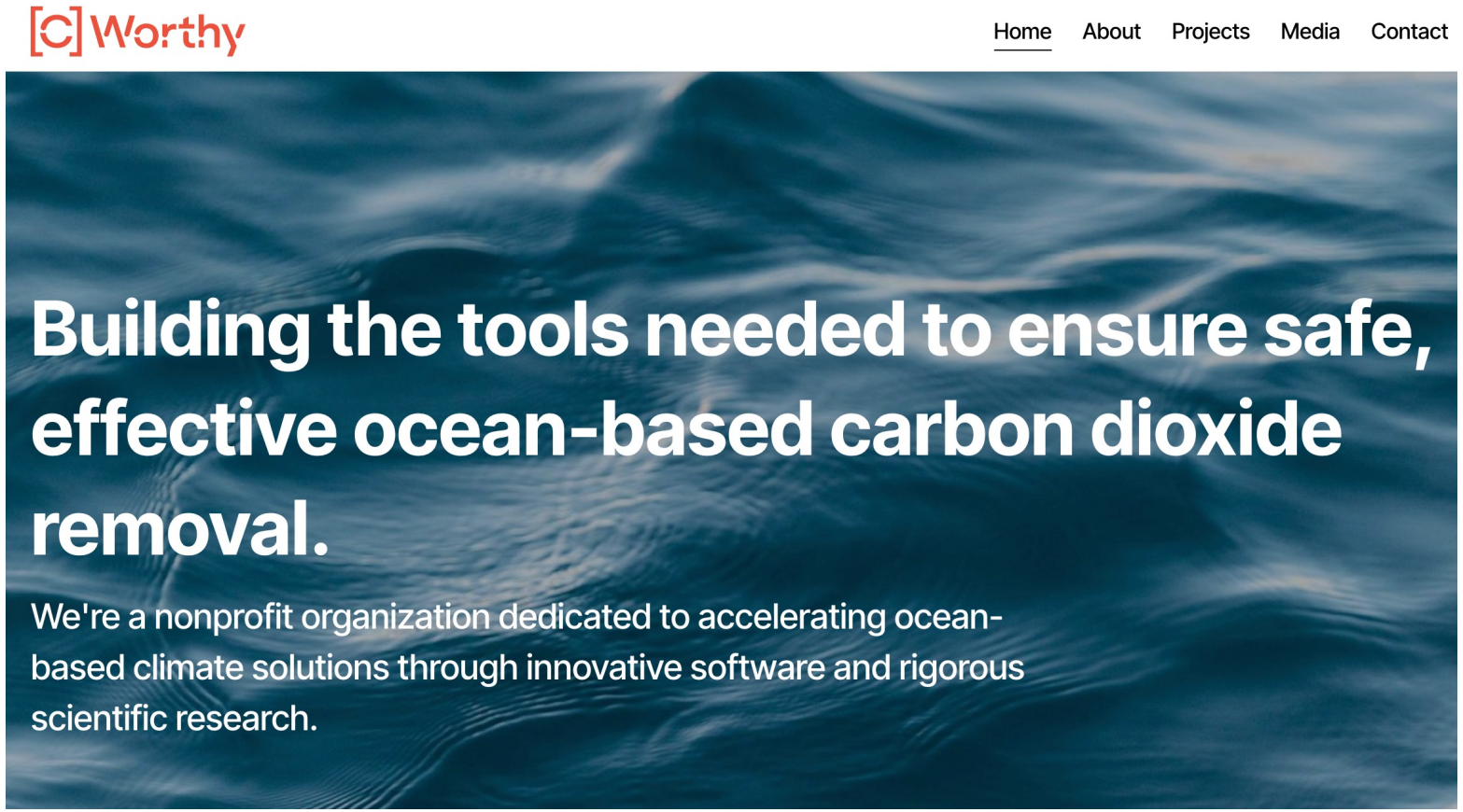
- Climate crisis continues to have widespread effects
- Scientists use computer modeling to study the potential impacts of different interventions such as ocean-based carbon dioxide removal (CDR)
- Need for scalable and high-resolution simulations of
  - Physical movement of tracers (CO2, NO3, PO4, NaCl, Fe, etc.) in the ocean
  - Biogeochemical (BGC) interactions between those tracers
- **The problem**: resolution mis-match between the physical simulation that determines tracer movement and interactions
- Execution time of a fully coupled BGC+physics model is an order of magnitude greater than the BGC interactions in isolation and the physical model offline

Carbon cycle processes, in gigatons per year



By Diagram adapted from U.S. DOE, Biological and Environmental Research Information System.
- http://earthobservatory.nasa.gov/Features/CarbonCycle/, Public Domain, https://commons.wikimedia.org/w/index.php?curid=19434238

[C]Worthy Experimenting with Approaches to Manage Resolution Mismatch



cworthy.org

- [C]Worthy is a nonprofit startup working to develop modeling and data to support ocean-based carbon dioxide removal (CDR)
- Experimenting with offline ocean movement and coarser granularity input to compute biogeochemical (BGC) interactions of tracers
- Prototyping distributed parallel CDR simulations using the Chapel programming language

The Chapel Parallel Programming Language

Chapel is a language designed for productive parallel programming, particularly on large-scale systems. Chapel is …

Easy to Use

"*We ask students at the master's degree to do stuff that would take 2 years and they do it in 3 months.*" Eric Laurendeau, Professor of Mechanical Engineering

Portable

HPE Cray EX, HPE Apollo, Cray XC, *nix systems, Mac, NVIDIA and AMD GPUs

Fast & Scalable

Achieved 8,500 GiB/s when sorting 256 TiB in 31 seconds on 8192 HPE Cray EX Nodes
… in just ~100 lines of Chapel

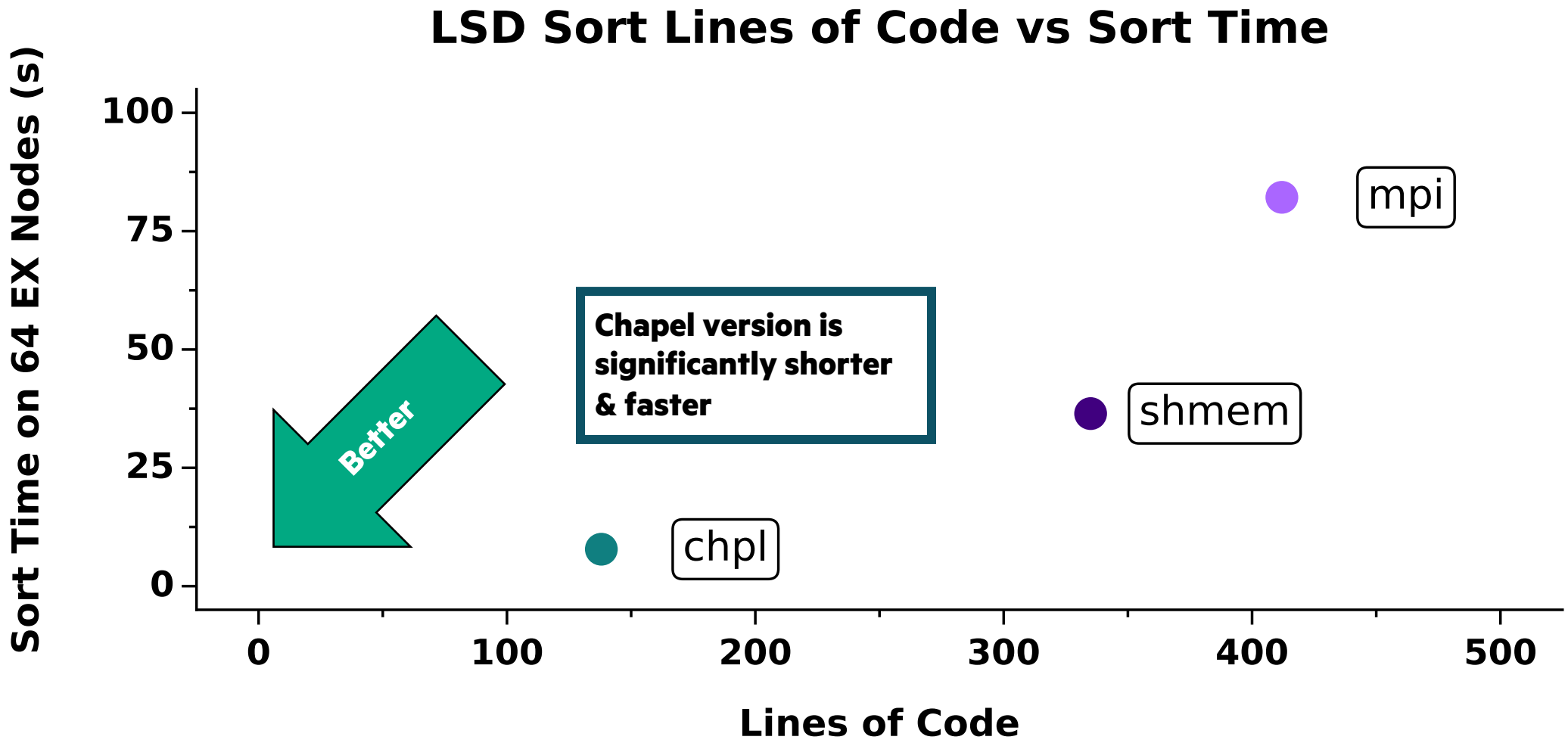
GPU-Ready

Real-world applications were ported on GPUs with few changes, and run on leadership-class systems such as Frontier and Perlmutter

Open source

Apache 2.0 Licensed, hosted on GitHub & accepted into HPSF
Learn more and engage with the Chapel community at chapel-lang.org

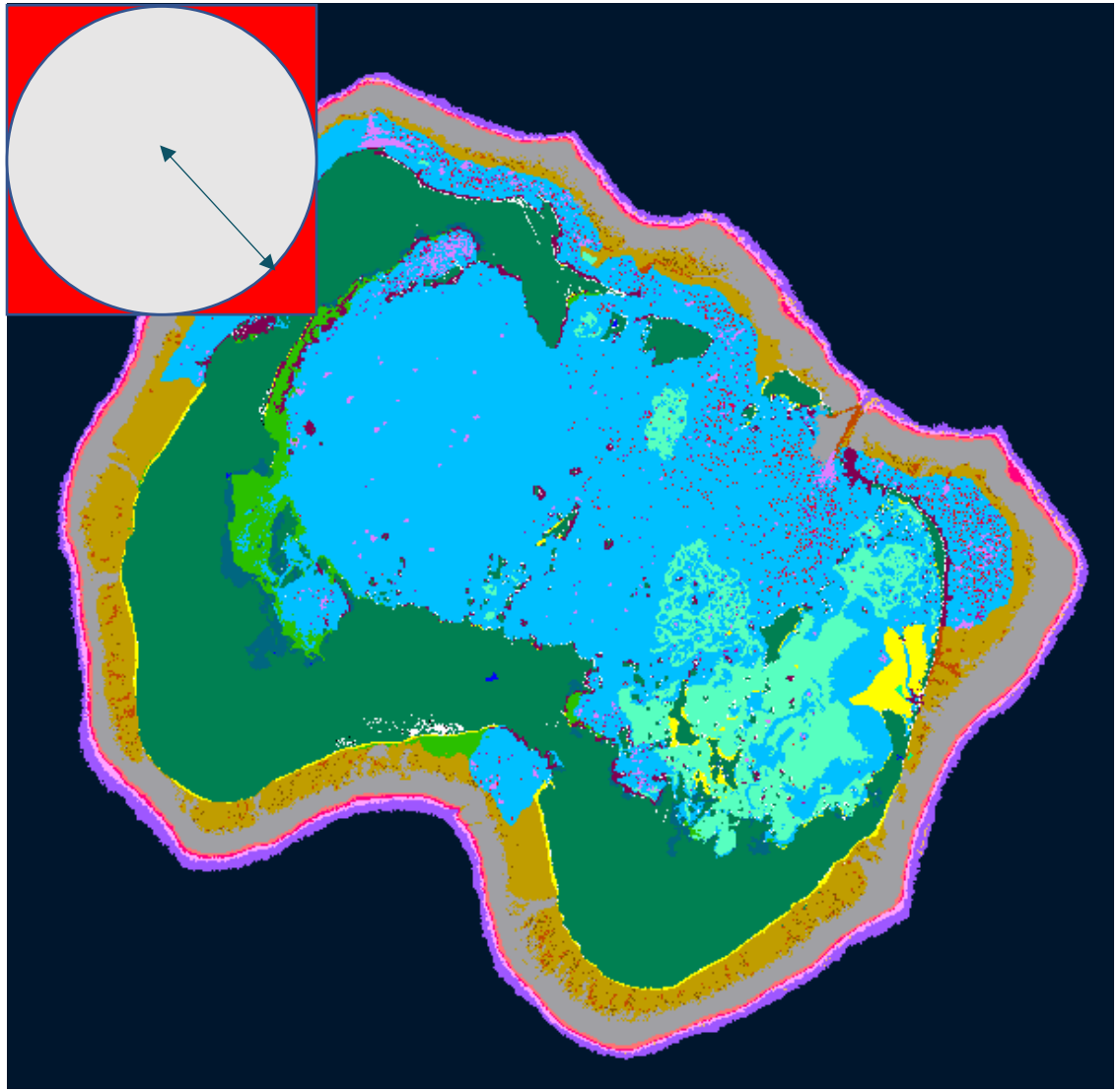
Sort Performance of 64 Billion Elements on 64 HPE Cray EX nodes



See https://github.com/mppf/distributed-lsb

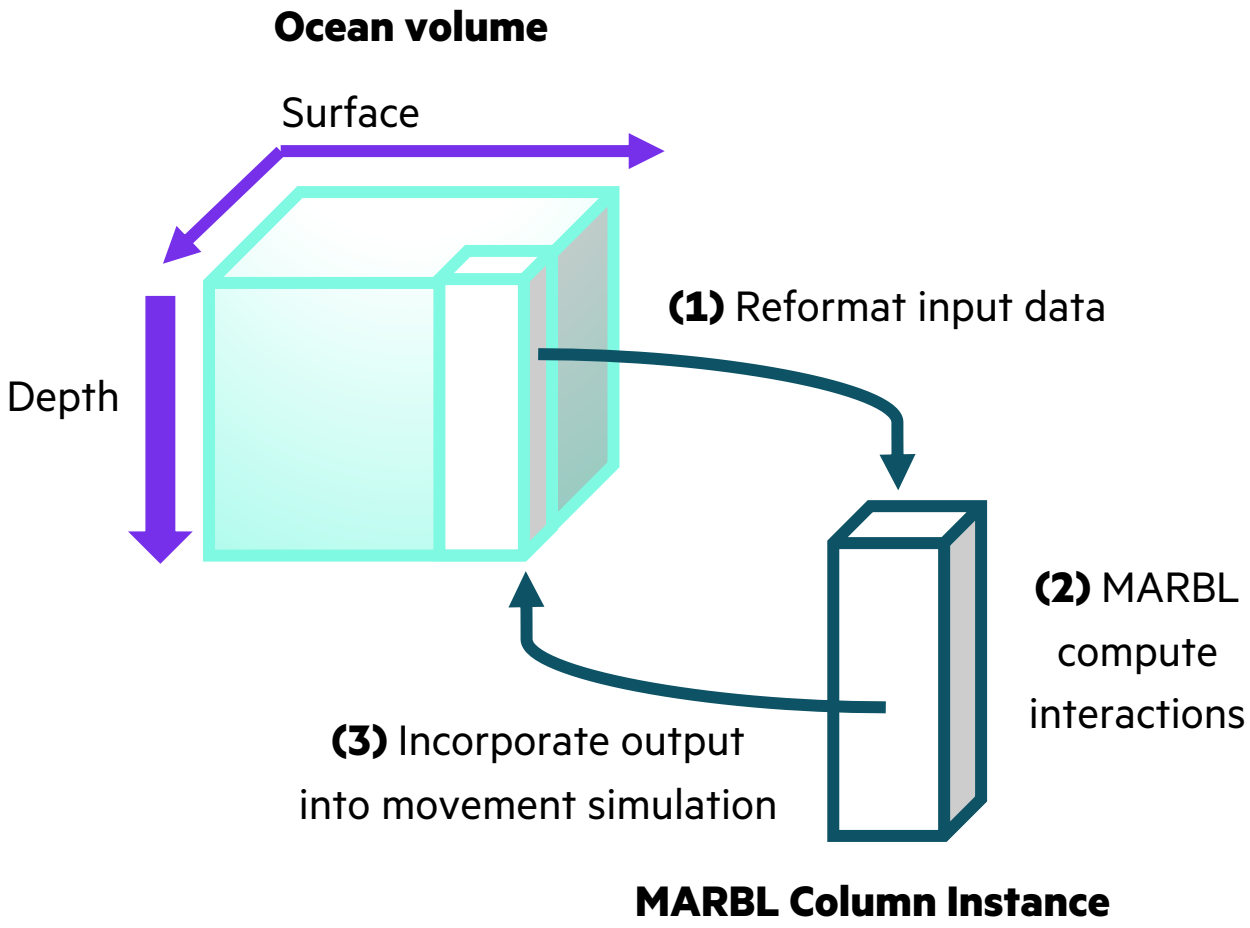
Scott's Motivation for Using Chapel

- **Analyzing images for coral reef diversity**
  - Important for prioritizing interventions
- **Algorithm implemented productively**
  - Add up weighted values of all points in a neighborhood
  - Developed by Scott Bachman, [C]Worthy tech lead
  - Scott …
    - started learning Chapel in Sept 2022,
    - started Coral Reef app in Dec 2022, and
    - had collaborators presenting results in Feb 2023
  - In July 2023, changed ~5 lines changed to run on GPU
- **Performance**
  - Fewer than 300 lines of Chapel code scales out to 100s of processors on Cheyenne (NCAR), also runs on Frontier
  - Full maps calculated in *seconds*, rather than days
    - 10,000 times faster than the MATLAB version



See Scott's CHIUW 2023 talk at https://youtu.be/lJhh9KLL2X0

Scott also wanted to use Marine Biogeochemistry Library (MARBL)



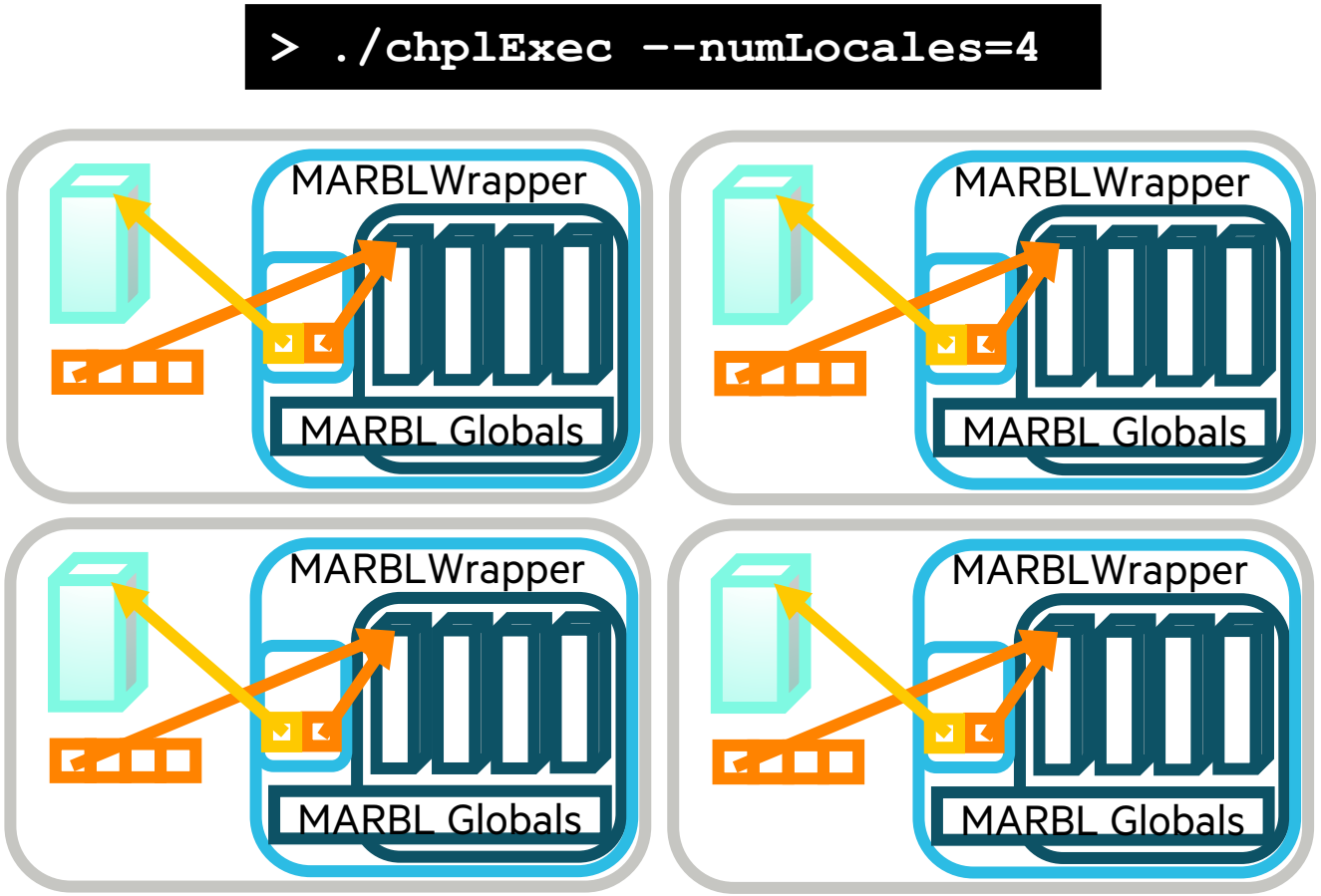
- MARBL, the **Mar**ine **B**iogeochemistry **L**ibrary, is a trusted Fortran library that simulates ocean biogeochemical processes, the interactions between the tracers (CO2, NO3, etc.)
- Used in several state-of-the-art global circulation models including: MOM6, MPAS, POP, and ROMS
  - Decomposes ocean into vertical columns with variable-height cells, ‘marbl_interface_class’
  - Each model like ROMS implements a Fortran driver/wrapper to exchange data with MARBL
- Nothing parallelized in MARBL, but each MARBL column instance treated independently

Calling MARBL Fortran Library from a Chapel Program

- Opportunity and Challenge
  - The ocean movement and parallelism can be rapidly prototyped in Chapel
  - The tracer interactions are in the MARBL Fortran library, and it doesn't make sense to rewrite them Chapel

- Solution
  - Correct parallel usage of MARBL is possible using C pointers for interoperability
  - Fortran wrapper is necessary for data exchange AND because MARBL column instances involve allocatable Fortran arrays

- Preliminary results show excellent weak and strong scaling on an HPE Cray EX, Perlmutter



Outline for the rest of the talk

Interoperability: Chapel and Fortran Working Together

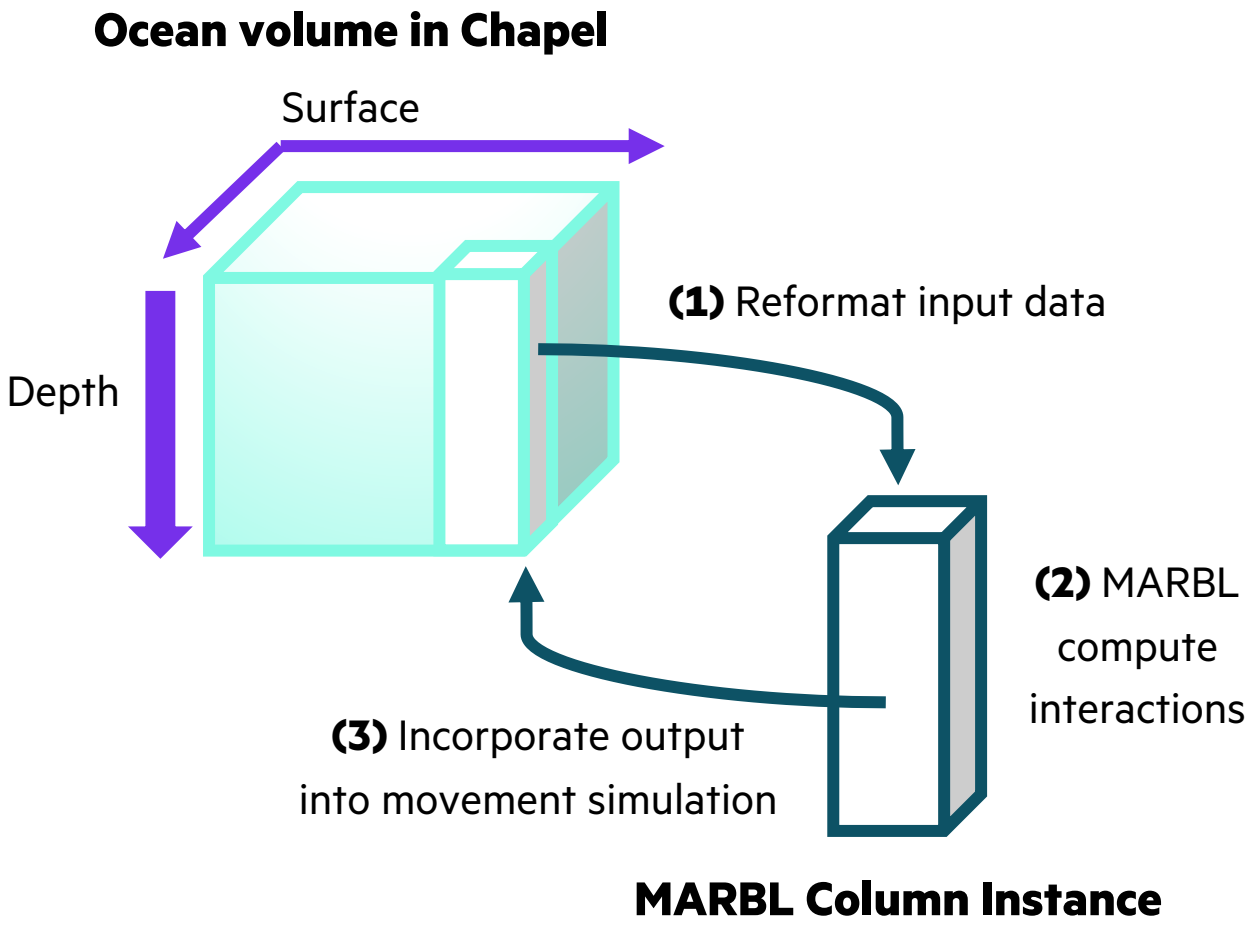
Using Chapel Distributed Arrays for Distributed and Shared Memory Parallelism

Preliminary Scaling Results on Perlmutter

Next Steps

Conclusion: Chapel Can Provide Parallelism to Trusted Fortran Libraries

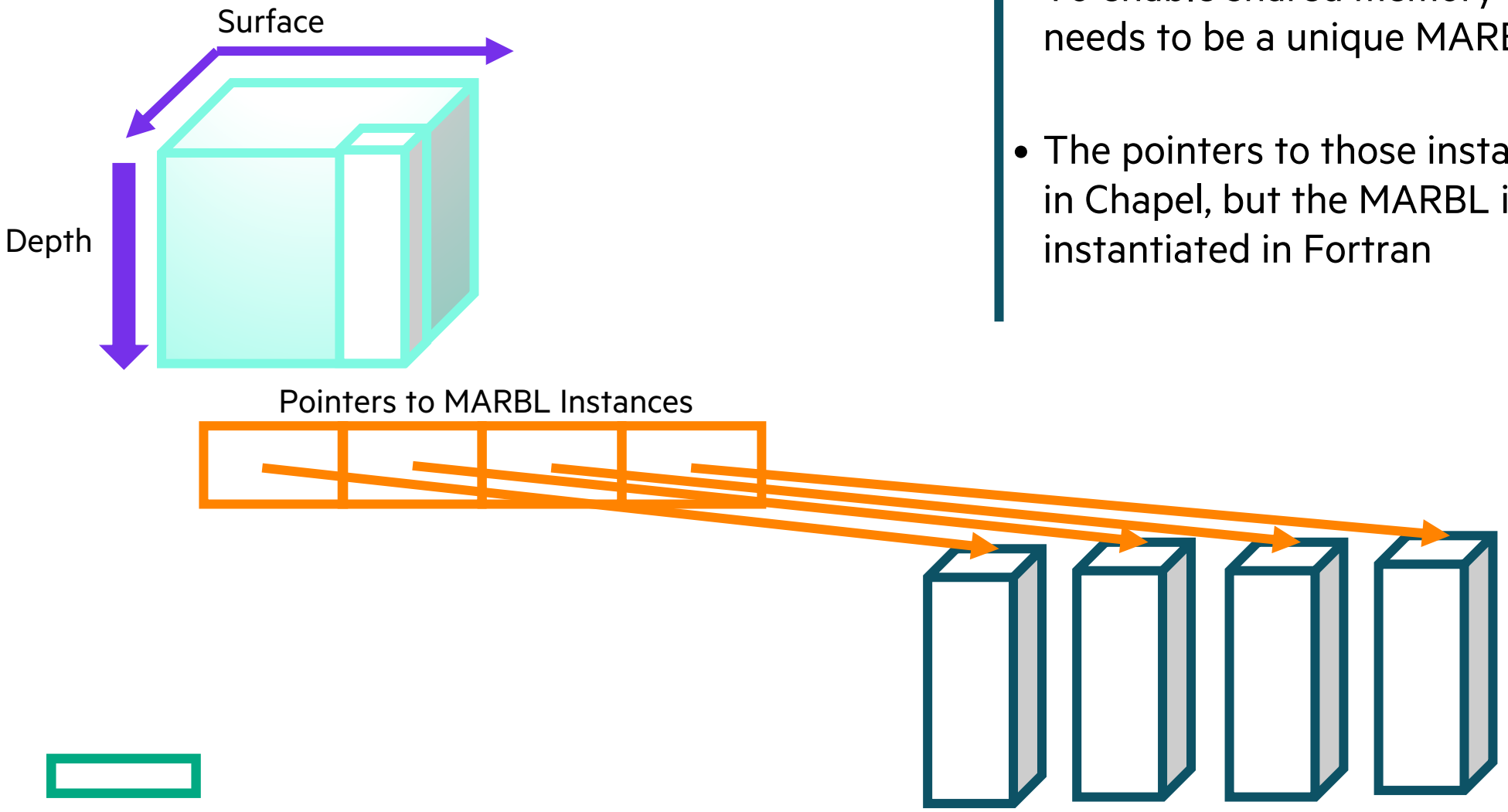
Interoperability Overview: Chapel and Fortran Working Together



- Chapel program calculates the tracer movement
  - This is where Scott is experimenting with an Offline Tracer Transport approach to avoid the order of magnitude cost of full simulation coupling
  - Each tracer can be transported independently
- MARBL simulates biogeochemical interactions
- As with other Global Circulation Models that use MARBL, there will be a Fortran wrapper for the data exchange between Chapel and MARBL

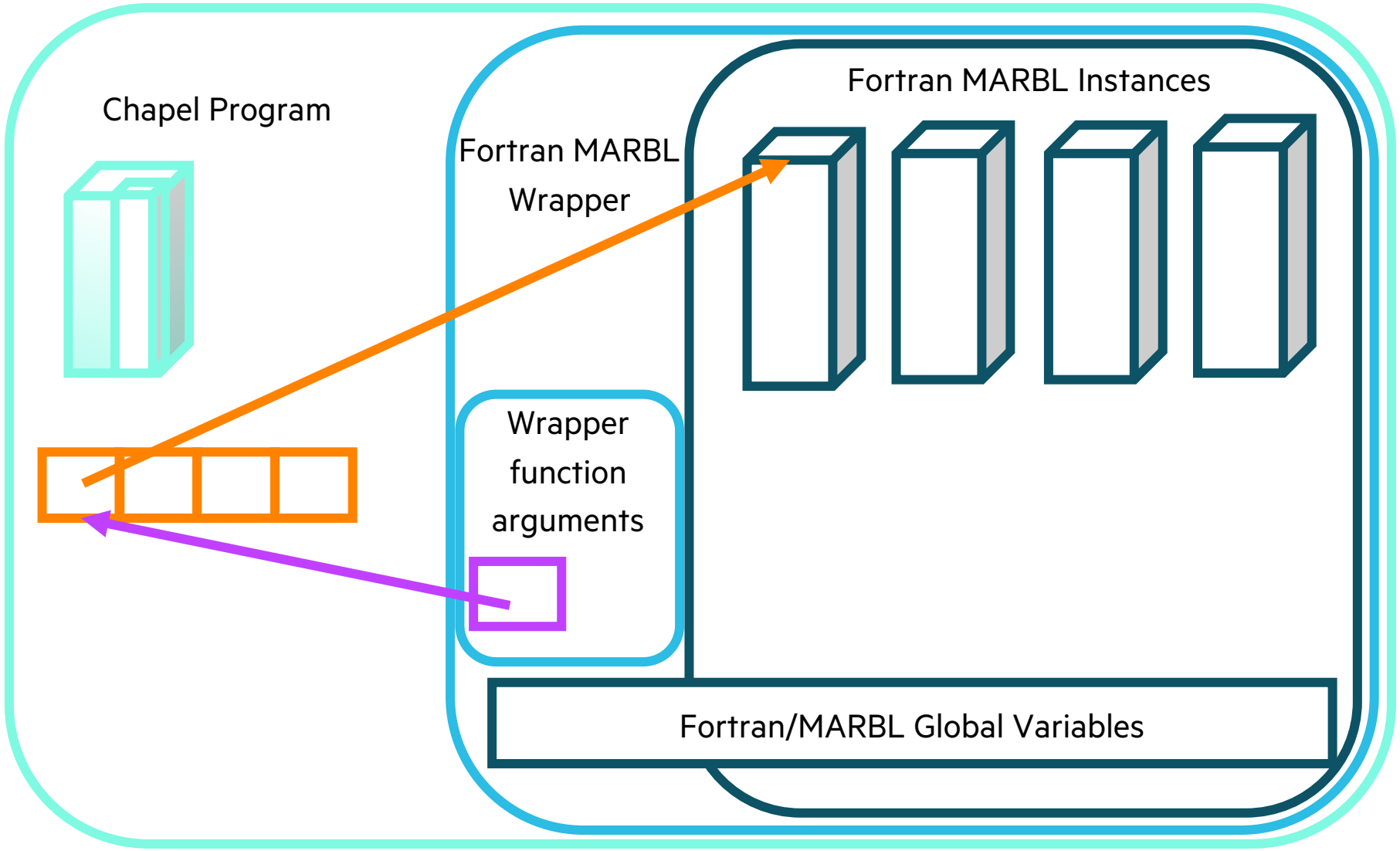
Shared Memory Parallelism

Need One MARBL Instance Per Thread/Task



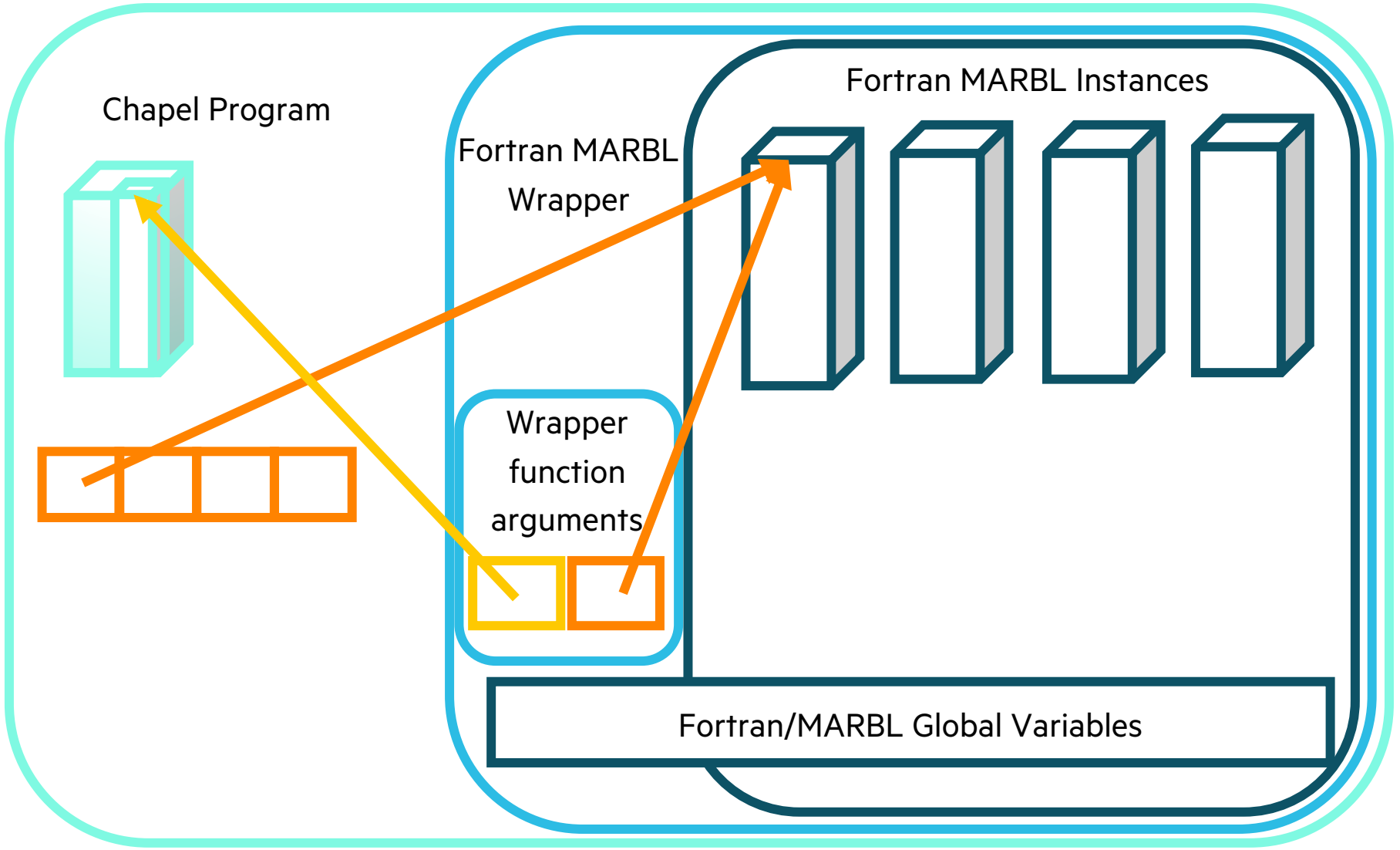
- To enable shared memory parallelism, there needs to be a unique MARBL instance per thread
- The pointers to those instances are maintained in Chapel, but the MARBL instances need to be instantiated in Fortran

Interoperability with C Pointers: Chapel and Fortran working together



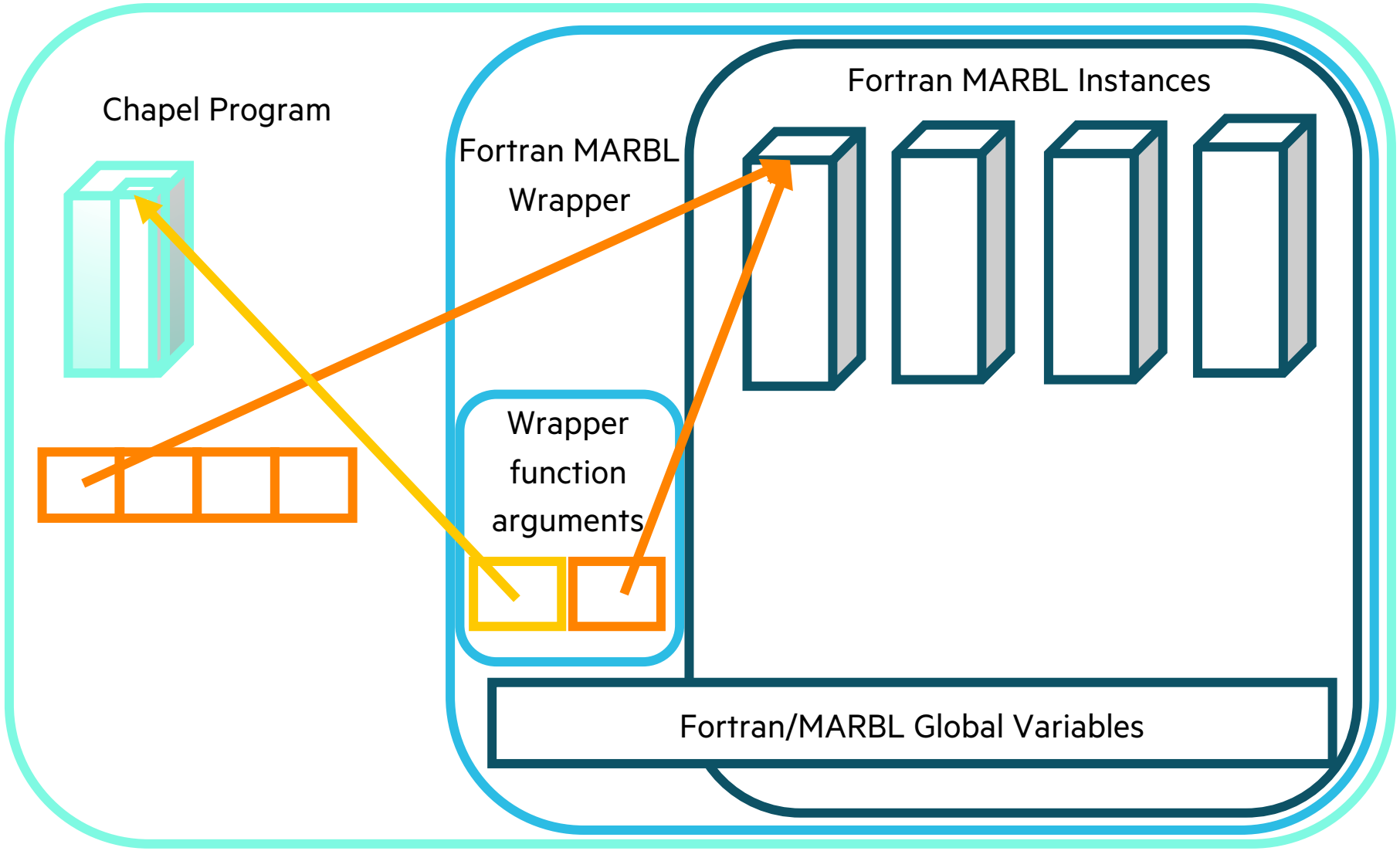
- Parallel Chapel code calls Fortran wrapper functions
- Wrapper function arguments are C pointers to portions of Chapel data and to Fortran MARBL instances
- Wrapper functions …
  0. Initialize MARBL instances
  1. Reformat data into MARBL
  2. Call MARBL computations
  3. Incorporate results back into Chapel
- Sidenote: MARBL has global variables accessible in Fortran Wrapper and MARBL

Interoperability with C Pointers: Chapel and Fortran working together



- Parallel Chapel code calls Fortran wrapper functions
- Wrapper function arguments are C pointers to portions of Chapel data and to Fortran MARBL instances
- Wrapper functions …
  0. Initialize MARBL instances
  1. Reformat data into MARBL
  2. Call MARBL computations
  3. Incorporate results back into Chapel
- Sidenote: MARBL has global variables accessible in Fortran Wrapper and MARBL

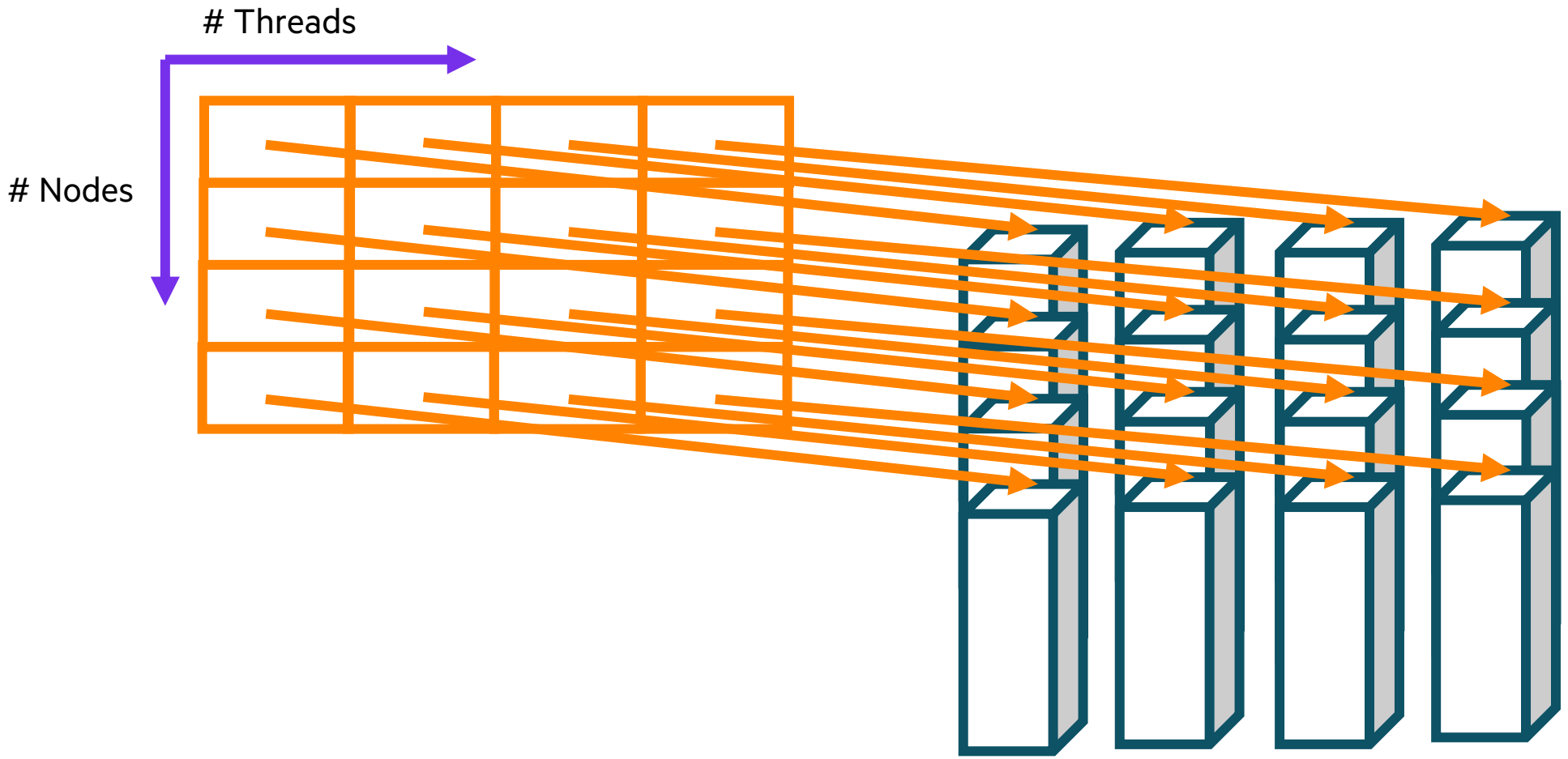
Interoperability with C Pointers: Chapel and Fortran working together



- Parallel Chapel code calls Fortran wrapper functions
- Wrapper function arguments are C pointers to portions of Chapel data and to Fortran MARBL instances
- Wrapper functions …
  0. Initialize MARBL instances
  1. Reformat data into MARBL
  2. Call MARBL computations
  3. Incorporate results back into Chapel
- Sidenote: MARBL has global variables accessible in Fortran Wrapper and MARBL

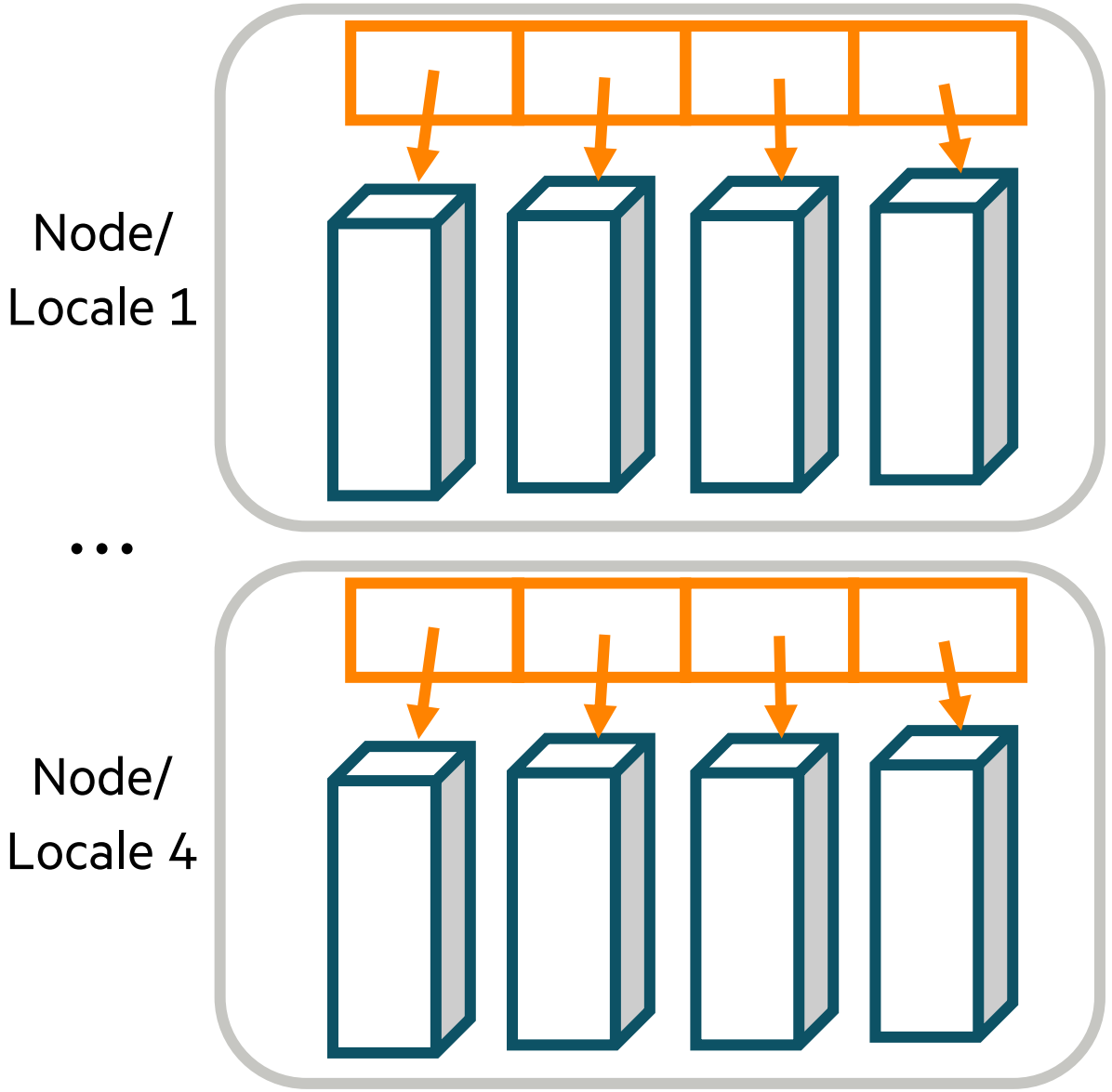
Distributed and Shared Memory Parallelism

Need One MARBL Instance Per Thread/Task AND Per Node



In Chapel, Declare the 2D Array of Handles as Distributed

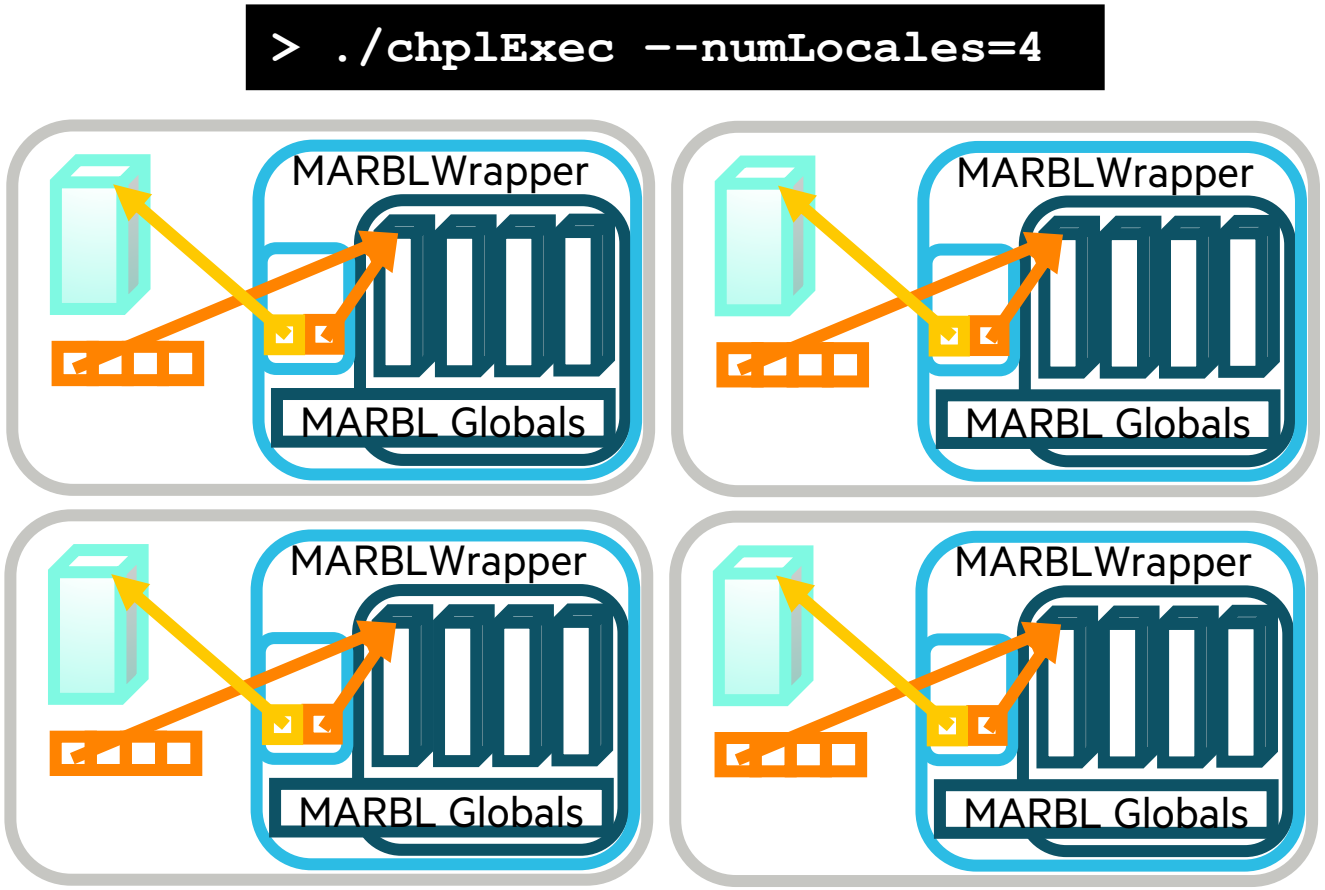
```
// Chapel code that declares a block distribution over
// the first dimension of the 2D array of MARBL instance handles.
use BlockDist;

const Dom = {1..Locales.size,1..nThreads};

const myTargetLocales =
  reshape(Locales,{1..Locales.size,1..1});

const DistRows =
  blockDist.createDomain(Dom, myTargetLocales);

// Declare array with distributed domain.
var HandleArray: [DistRows] MarbleHandle;
```



Distributed Parallel Chapel Program Using MARBL Fortran Library

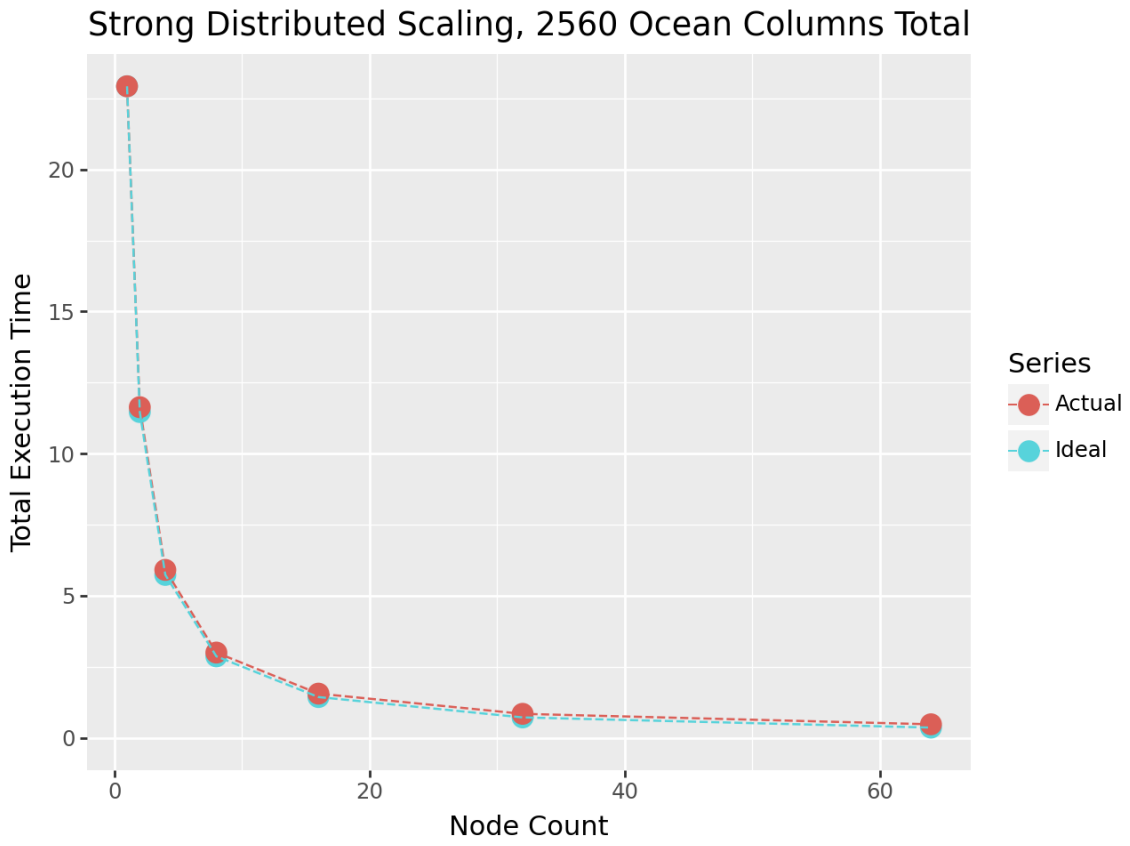
- The Chapel program distributes the ocean data array and the array of MARBL pointers
- As shown, the executable will launch a Chapel locale (i.e., rank) per node for 4 nodes
- ‘forall’ loops over distributed arrays results in distributed and shared memory parallelism

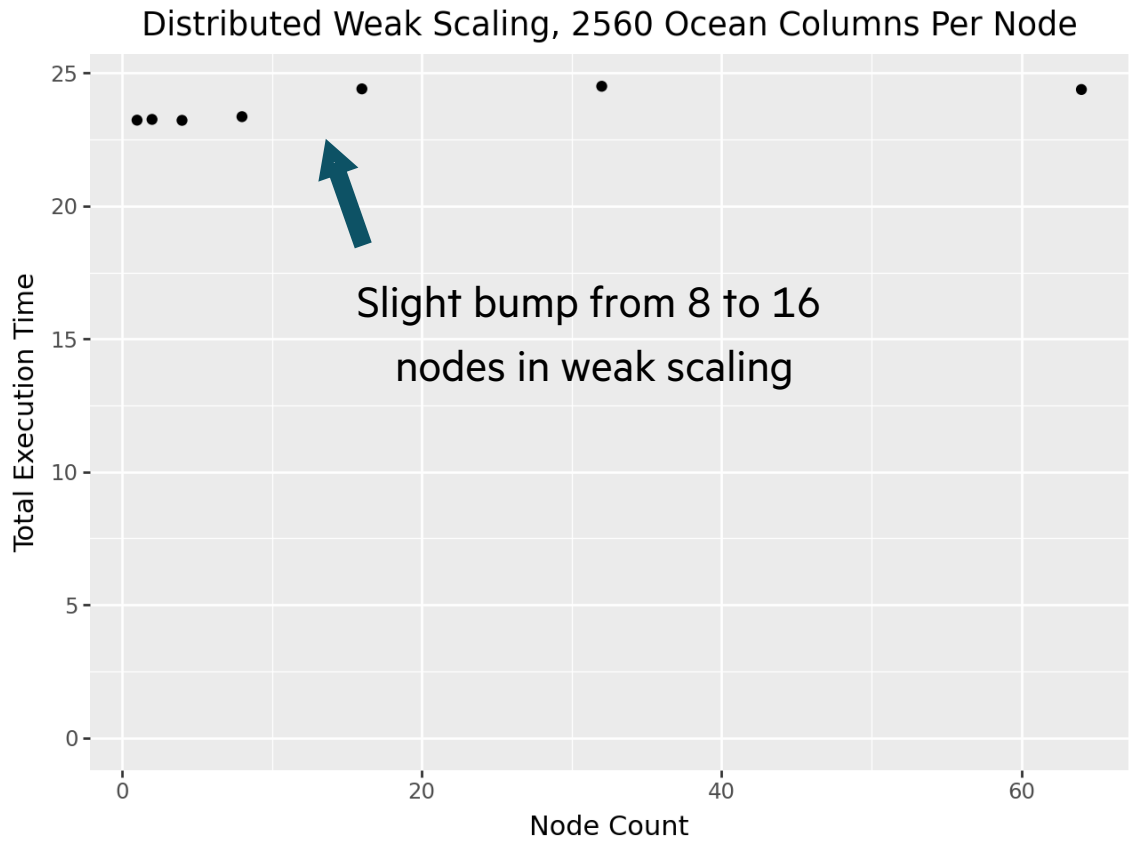
```
forall (loc,task) in DistRows {

      // computation for each parallel task on each locale

}
```



Evaluation: Near Perfect Distributed Parallel Scaling on Perlmutter

As expected, due to lack of interactions between columns





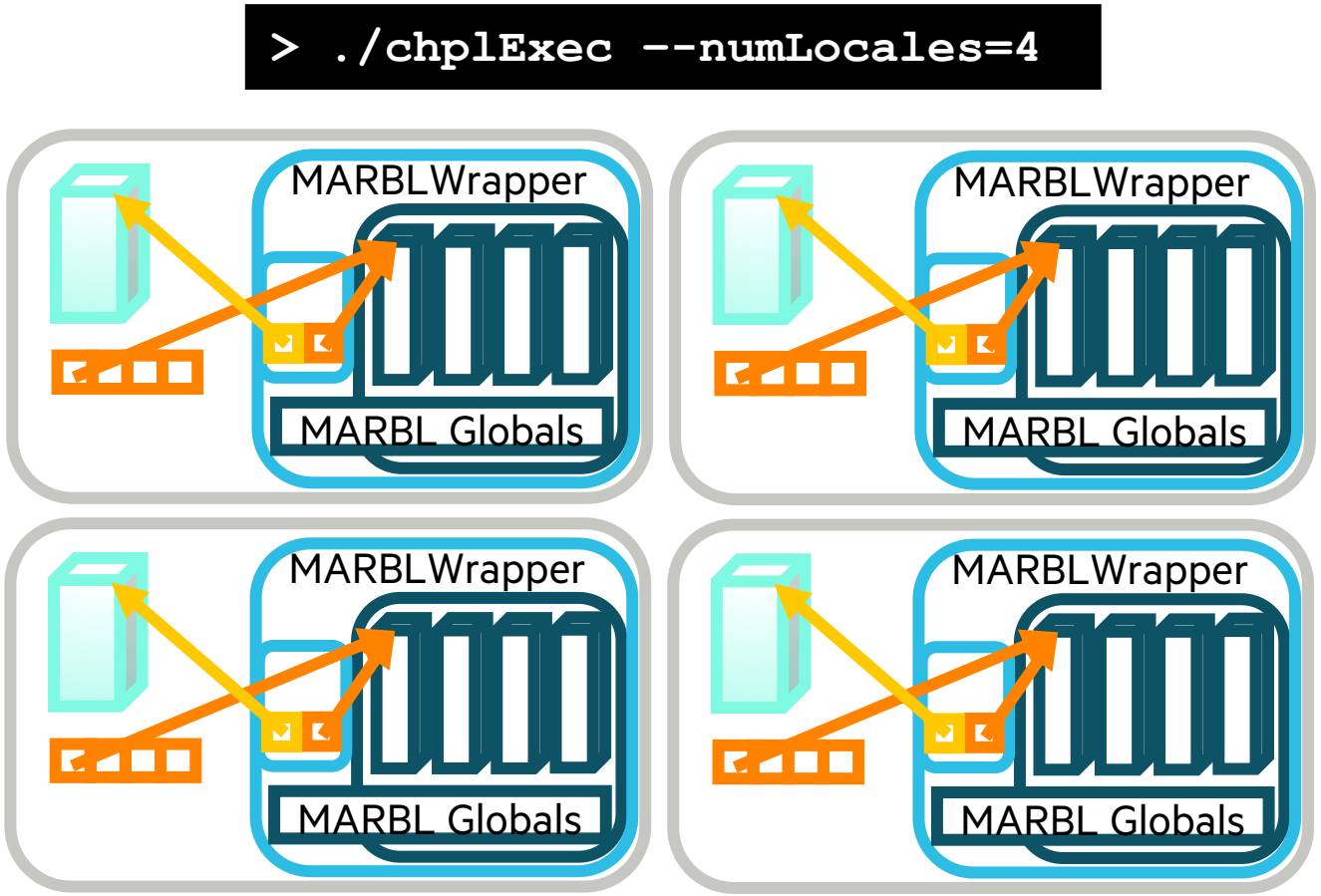
Next Steps for Carbon Dioxide Removal Simulation

- Incorporating the Fortran wrapper into the Offline Tracer Transport code Scott is writing
- Investigate limits to grid coarsening: how far can [C]Worthy take the Offline Tracer Transport approach without losing accuracy?
- Performance tune data exchanges between Chapel and Fortran data structures

Credit: https://www.cworthy.org/media

Conclusion: Chapel Can Provide Parallelism to Trusted Fortran Libraries

- **Chapel provides …**
  - Rapid prototyping of alternative approaches for tracer movement in the ocean
  - Distributed and shared memory parallelism
- **Fortran Library MARBL provides …**
  - Biogeochemical interaction simulation for tracers
  - Trust from use in other circulation models
- **Scaling and Performance**
  - Scaling well on Perlmutter
  - Data exchange tuning is needed but possible
- **Blog Post and Contact Info**
  - See https://chapel-lang.org/blog/posts/fortran-marbl1/ for more details on Chapel/Fortran interop
  - brandon.neth@hpe.com, scott@cworthy.org, michelle.strout@hpe.com

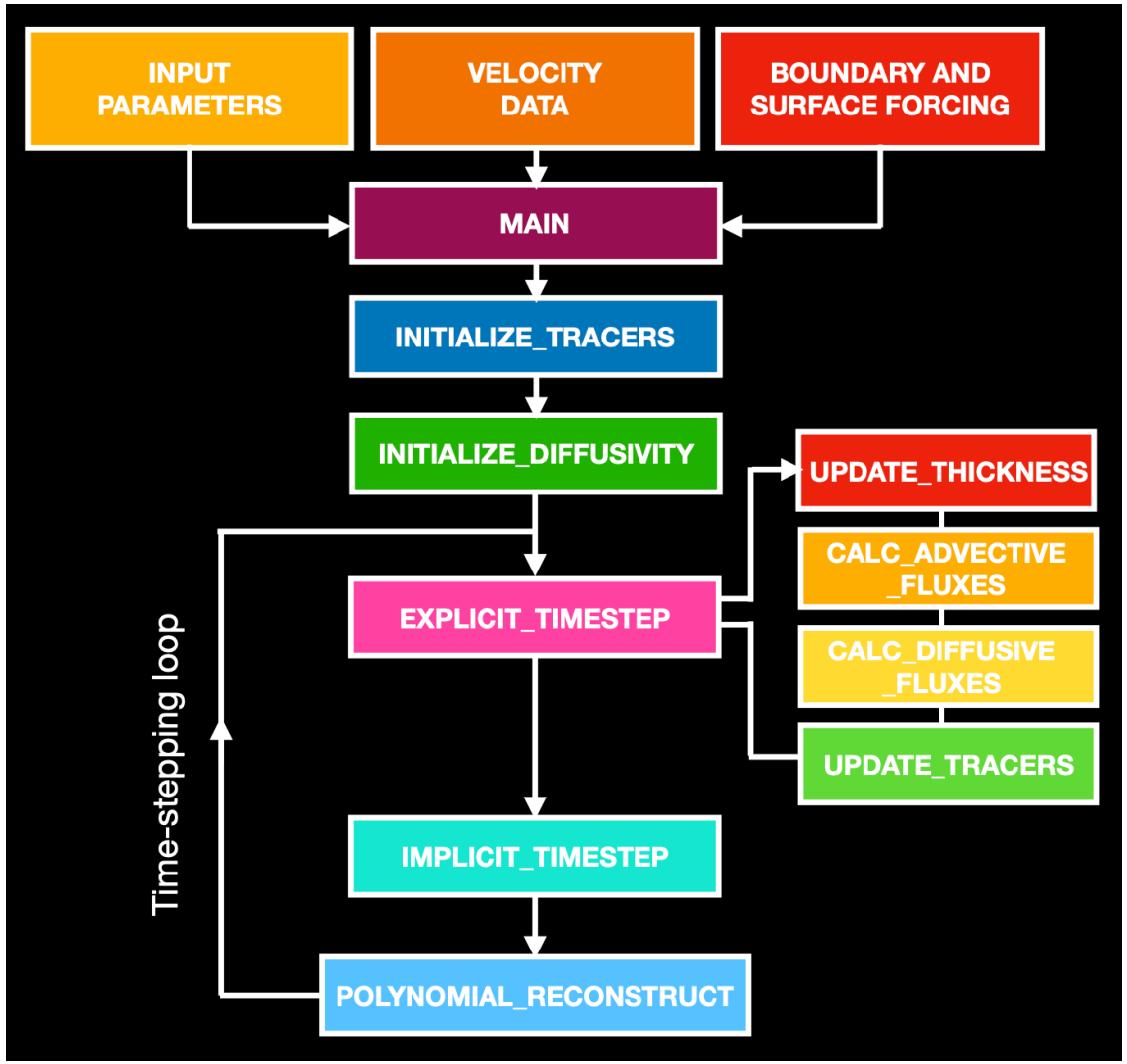


Thank you

michelle.strout@hpe.com

Backup Slides

Offline Tracer Transport Implemented in Chapel That Calls MARBL Library



Courtesy of Scott Bachman, [C]Worthy

- Step 1: Read input data, including pre-calculated velocity data
- Step 2: Initialize tracers (concentrations of different compounds) and diffusivity (how quickly different compounds diffuse through the ocean)
- Step 3: Begin stepping
  - 3.1: Update geometry based on flow
  - 3.2: Advective fluxes: Changes in concentration due to fluid flow (pre-calculated velocities)
  - 3.3: Diffusive fluxes: Changes in concentration due to diffusion
  - **3.4: Update tracers: BGC calculations using MARBL**
  - 3.5: Vertical diffusion (implicit timestep)
  - 3.6: Polynomial interpolation for regridding

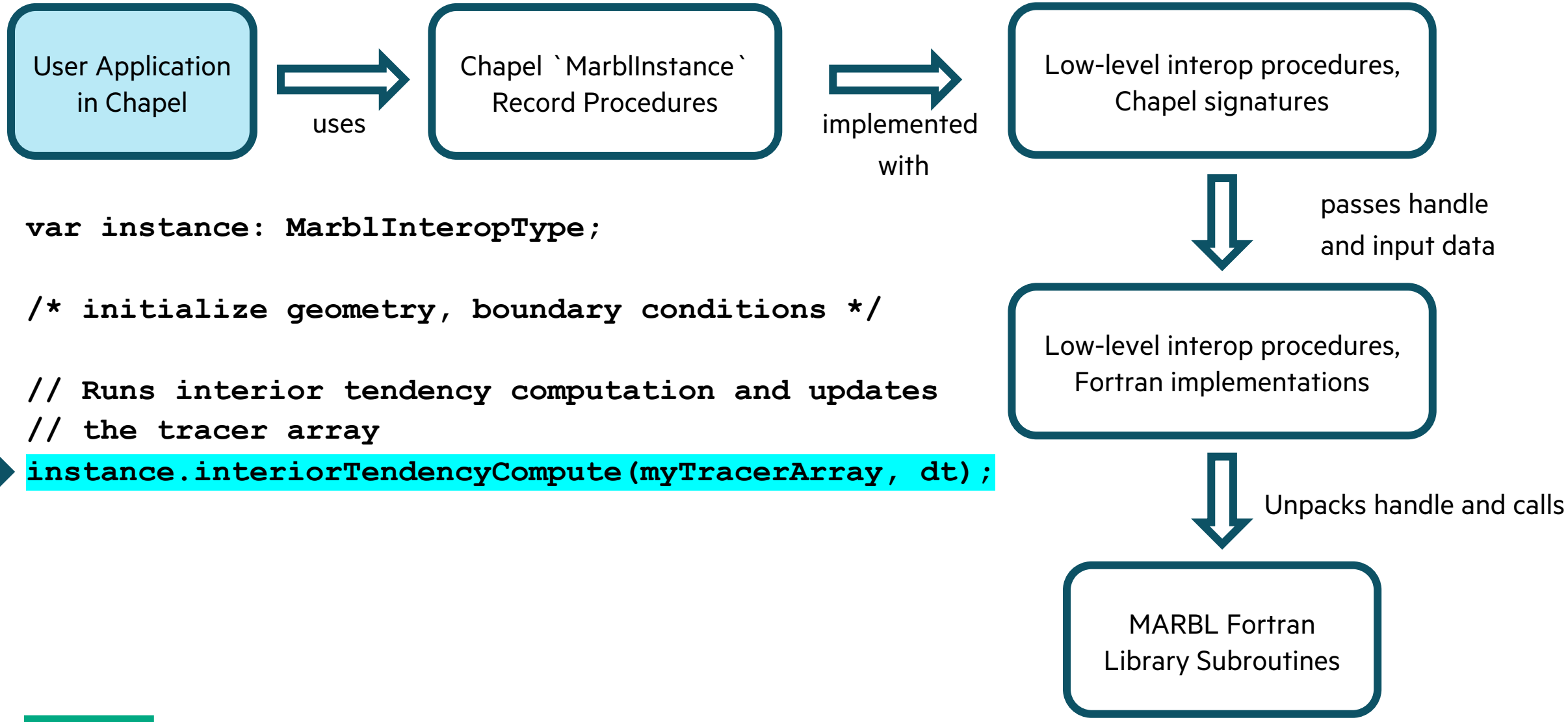
Challenge : Limited Fortran Interoperability Support

- Fortran does not support interoperability for derived data types (user-defined data structures) that use `allocatable` arrays, which are present in MARBL's key `marbl_instance_class` data structure
- Thus, cannot pass `marbl_instance_class` objects directly between Fortran and Chapel code
- However, Fortran *does* support interoperability for its `c_ptr` type

- Approach: Use a `c_ptr` to the `marbl_instance_class` as a handle between the Chapel and Fortran side of the interoperability layer
  - Chapel-side user application creates a `MarblInteropType` object, which contains a `void*` C pointer field
  - Chapel-side procedures accept `MarblInteropType` as an argument
  - Fortran-side implementation converts the opaque `c_ptr` field to a usable `marbl_instance_class` object, then calls the relevant MARBL subroutine

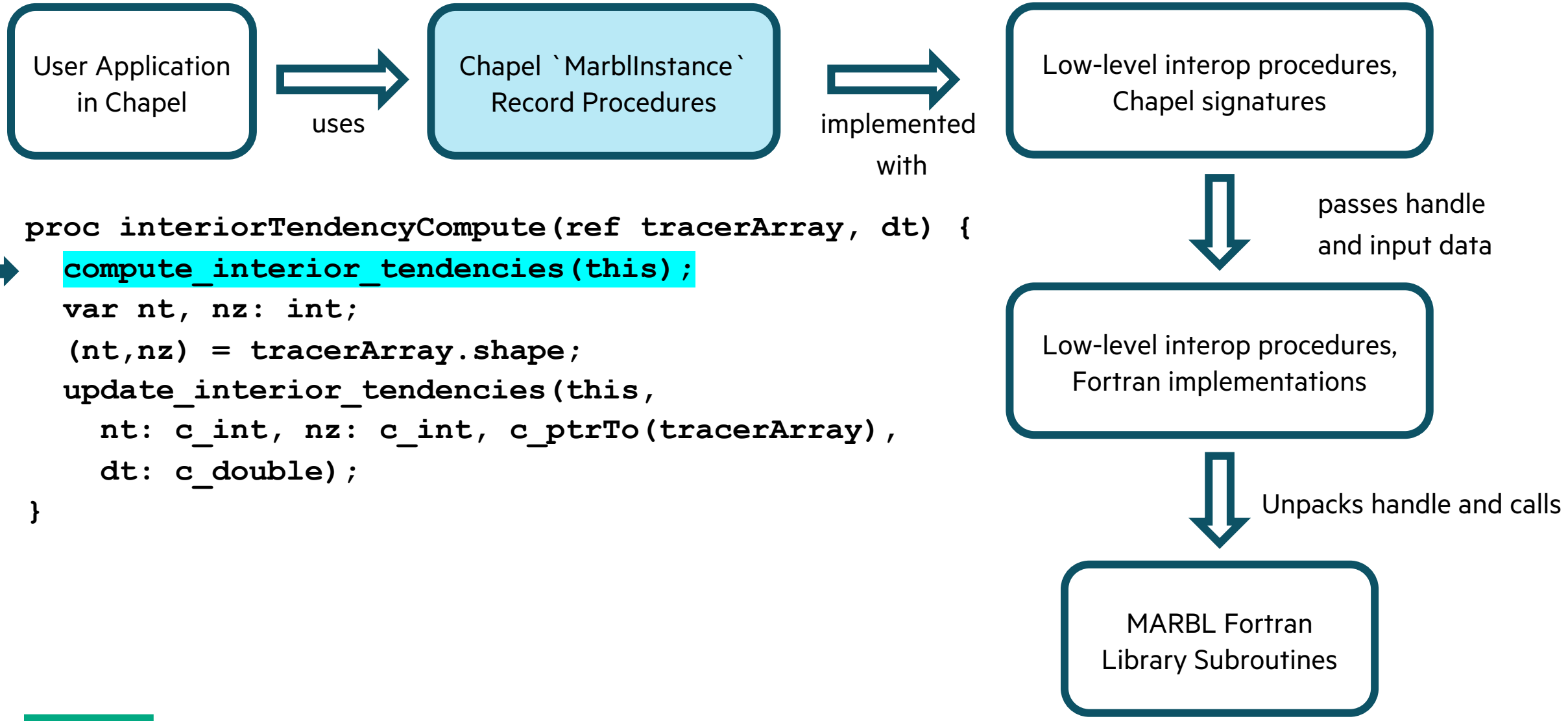
- Bonus: The approach of having code on both sides of the language barrier side-steps the complexity of setting up the MARBL data structure in the Chapel application
  - Can pass Chapel arrays to interop procedures that populate the `marbl_instance_class` fields

Chapel/Fortran Interoperability: Stepping through an example



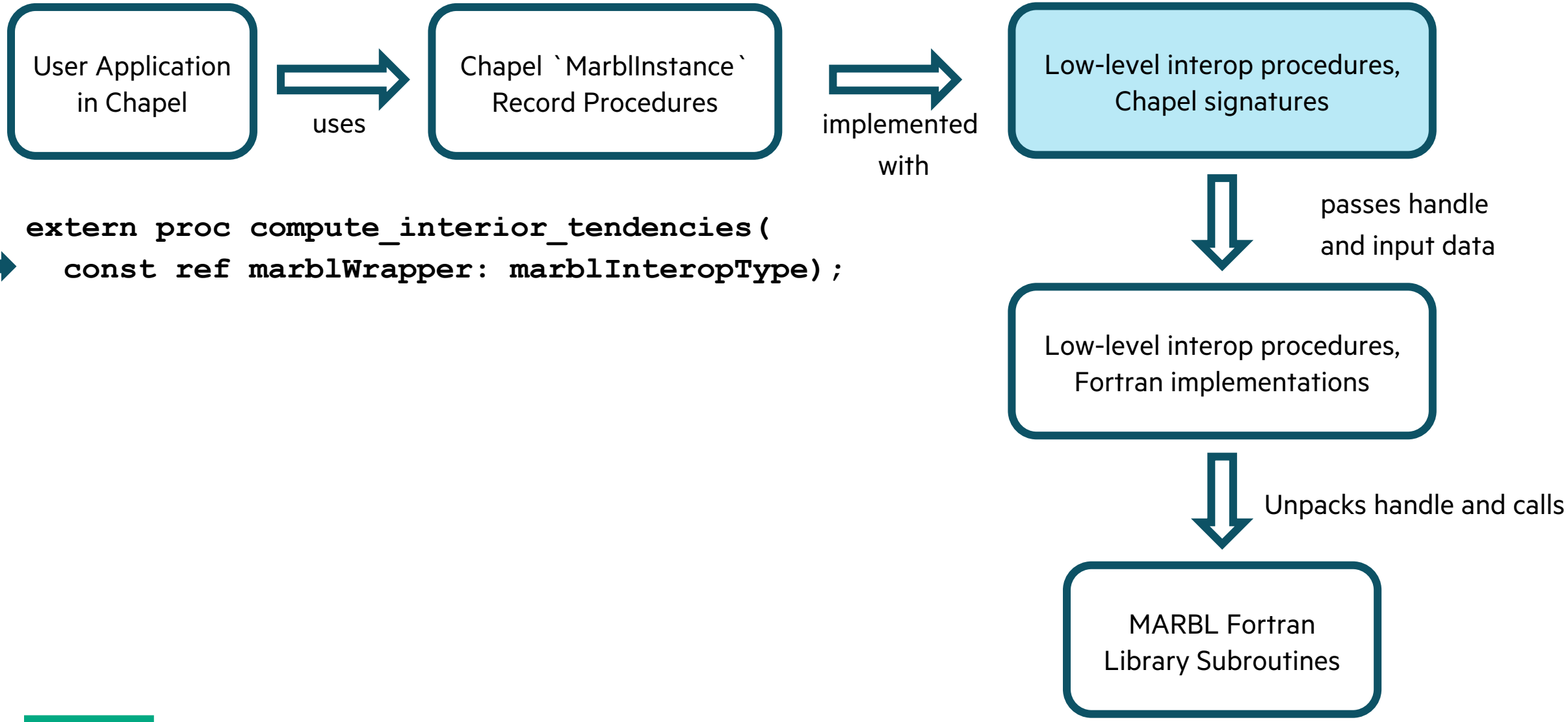

```
var instance: MarblInteropType;

/* initialize geometry, boundary conditions */

// Runs interior tendency computation and updates
// the tracer array
instance.interiorTendencyCompute(myTracerArray, dt);
```

Chapel MarblInstance data structure calls into Fortran wrapper for MARBL



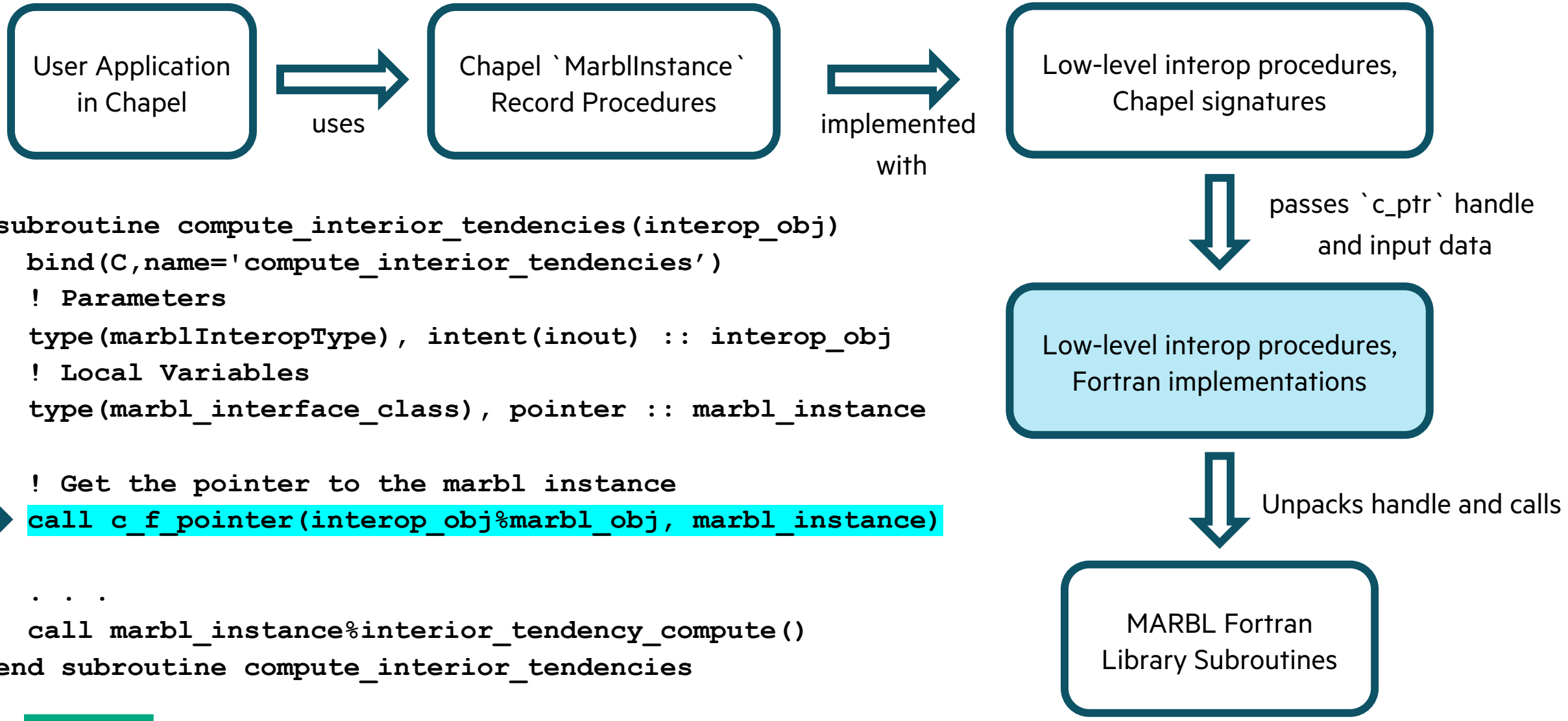

```
proc interiorTendencyCompute(ref tracerArray, dt) {
  compute_interior_tendencies(this);
  var nt, nz: int;
  (nt,nz) = tracerArray.shape;
  update_interior_tendencies(this,
    nt: c_int, nz: c_int, c_ptrTo(tracerArray),
    dt: c_double);
}
```

Chapel Extern Signature indicates a c_ptr to Fortran Wrapper



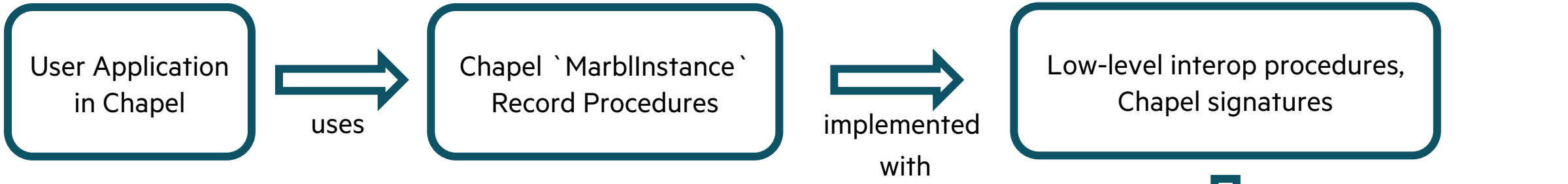

```
extern proc compute_interior_tendencies(
  const ref marblWrapper: marblInteropType);
```

FORTRAN Wrapper code pulls C pointer to marbl_instance out of Chapel record




```
subroutine compute_interior_tendencies(interop_obj)
  bind(C,name='compute_interior_tendencies')
  ! Parameters
  type(marblInteropType), intent(inout) :: interop_obj
  ! Local Variables
  type(marbl_interface_class), pointer :: marbl_instance

  ! Get the pointer to the marbl instance
  call c_f_pointer(interop_obj%marbl_obj, marbl_instance)

  . . .
  call marbl_instance%interior_tendency_compute()
end subroutine compute_interior_tendencies
```

FORTRAN Wrapper then calls compute method on Fortran marbl_instance




```
subroutine compute_interior_tendencies(interop_obj)
  bind(C,name='compute_interior_tendencies')
  ! Parameters
  type(marblInteropType), intent(inout) :: interop_obj
  ! Local Variables
  type(marbl_interface_class), pointer :: marbl_instance

  ! Get the pointer to the marbl instance
  call c_f_pointer(interop_obj%marbl_obj, marbl_instance)

  . . .
  call marbl_instance%interior_tendency_compute()
end subroutine compute_interior_tendencies
```