OAK RIDGE National Laboratory

Using OpenMP to Harness GPUs for Core-Collapse Supernova Simulations with GenASiS

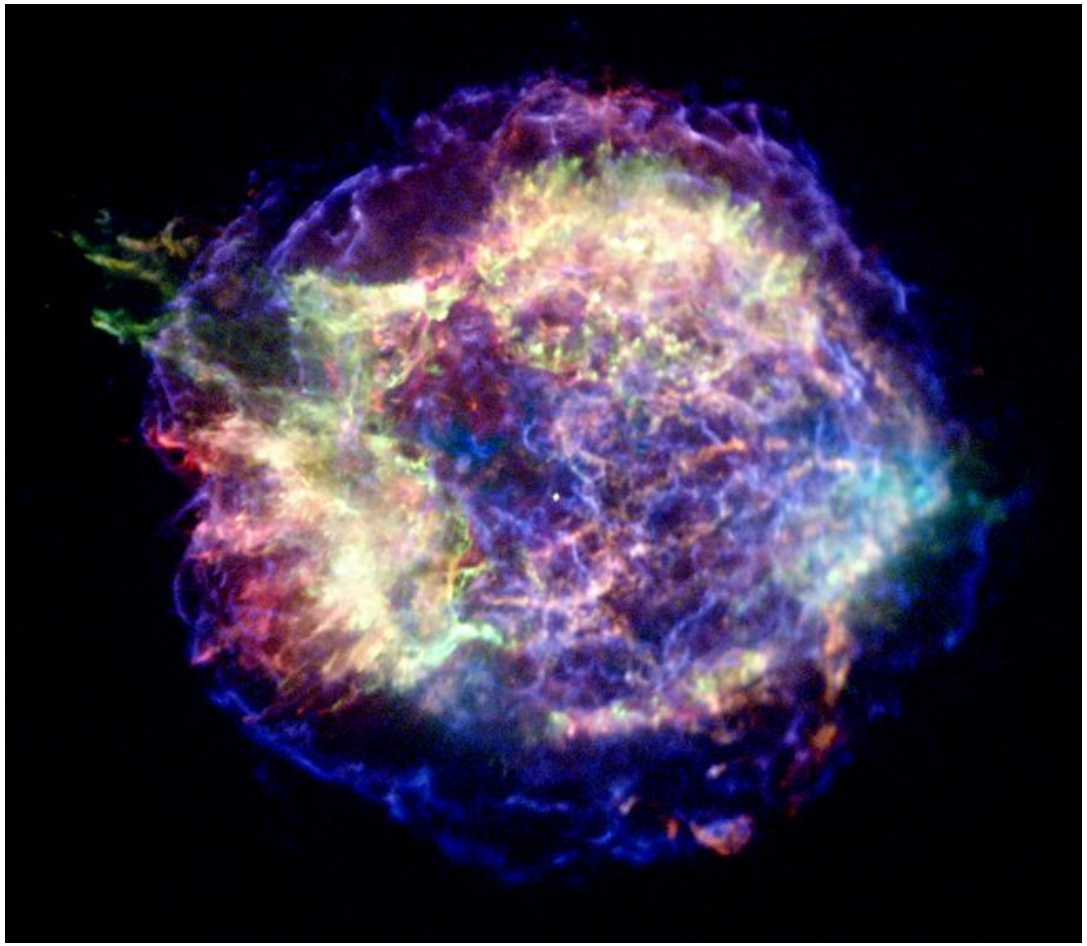
Reuben D. Budiardja
Computational Scientist
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

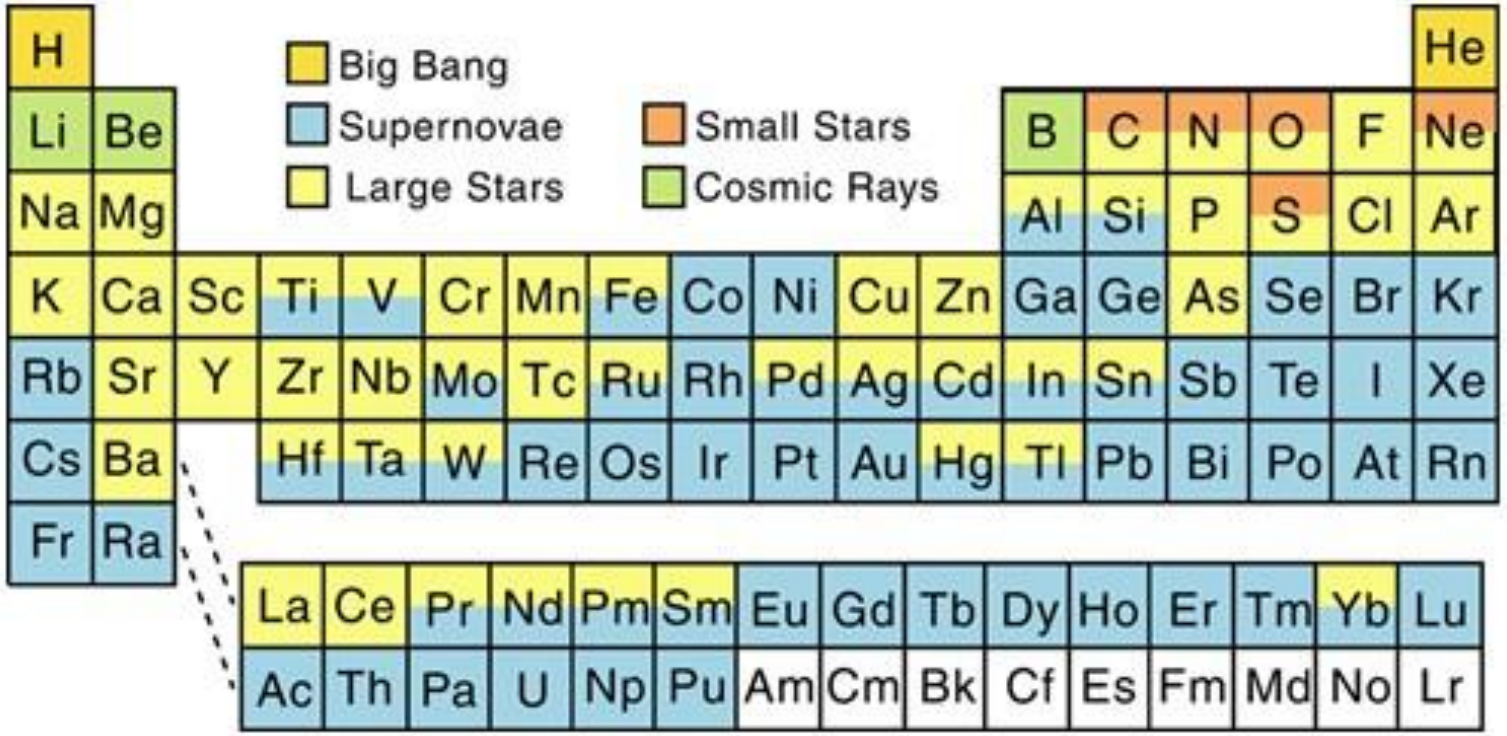
Core-Collapse Supernovae (CCSN)

- The death throes of massive star (M > ~10 Solar M)
  - The birth of neutron stars and black holes
- Among the most powerful explosions in the universe
  - ~10^{53} ergs of energy released as mostly neutrino
  - ~10^{51} ergs visible electromagnetic radiation
    ~ 10^{28} megatonnes of TNT
- Observables:
  - Gamma-ray burst, gravitational wave, neutrino
- Occur about twice per century in our galaxy

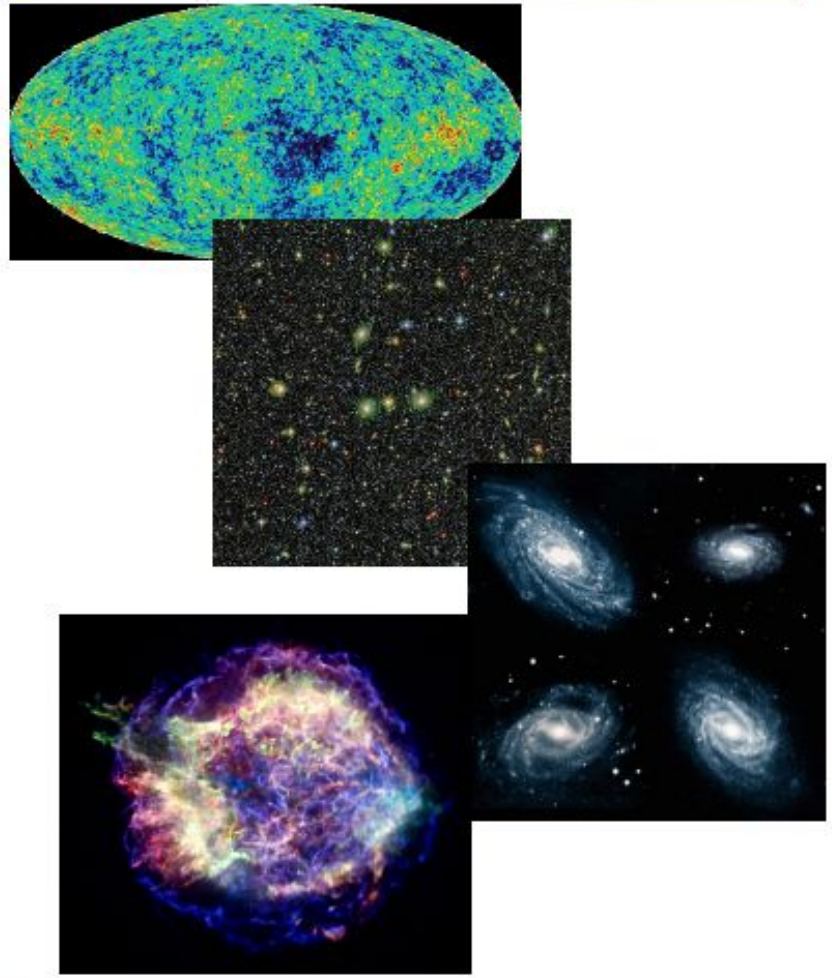


Cassiopeia A Supernova Remnant (Chandra Observatory)

Why Simulate Supernova?

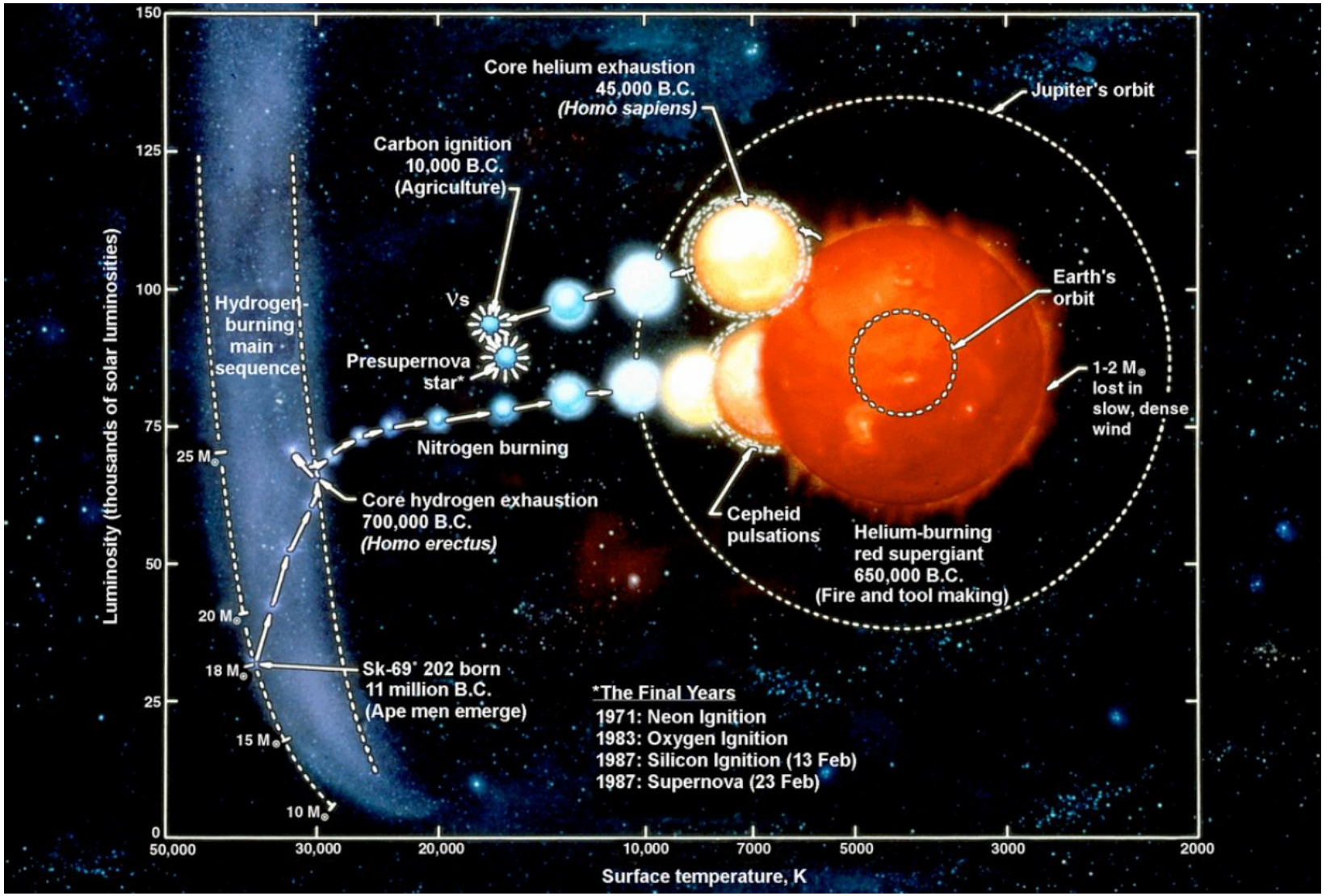


Understanding our universe and our place in it will require an understanding of phenomena on all scales.



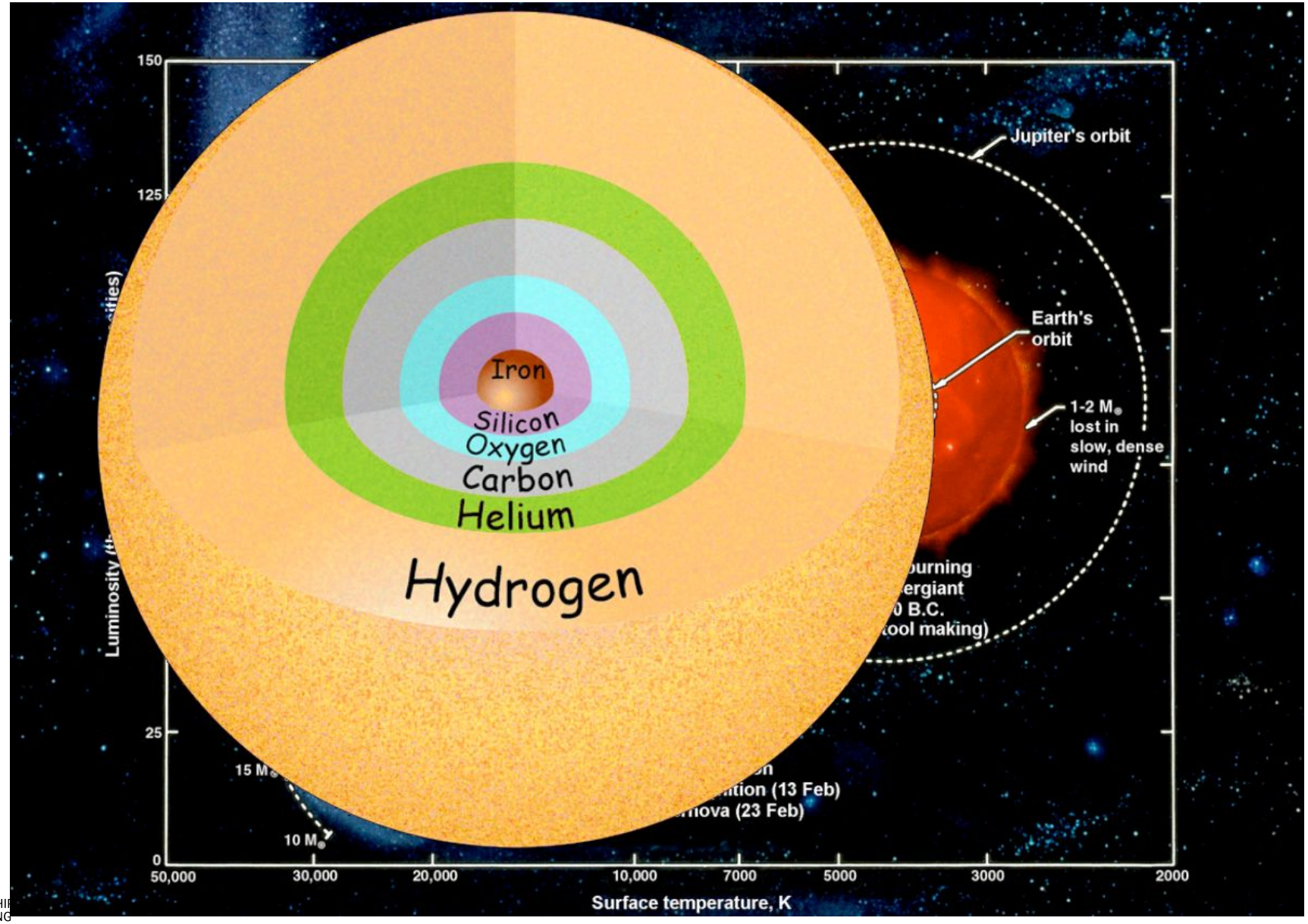
Answers to questions relating to our origins in the universe

The Path to Explosion

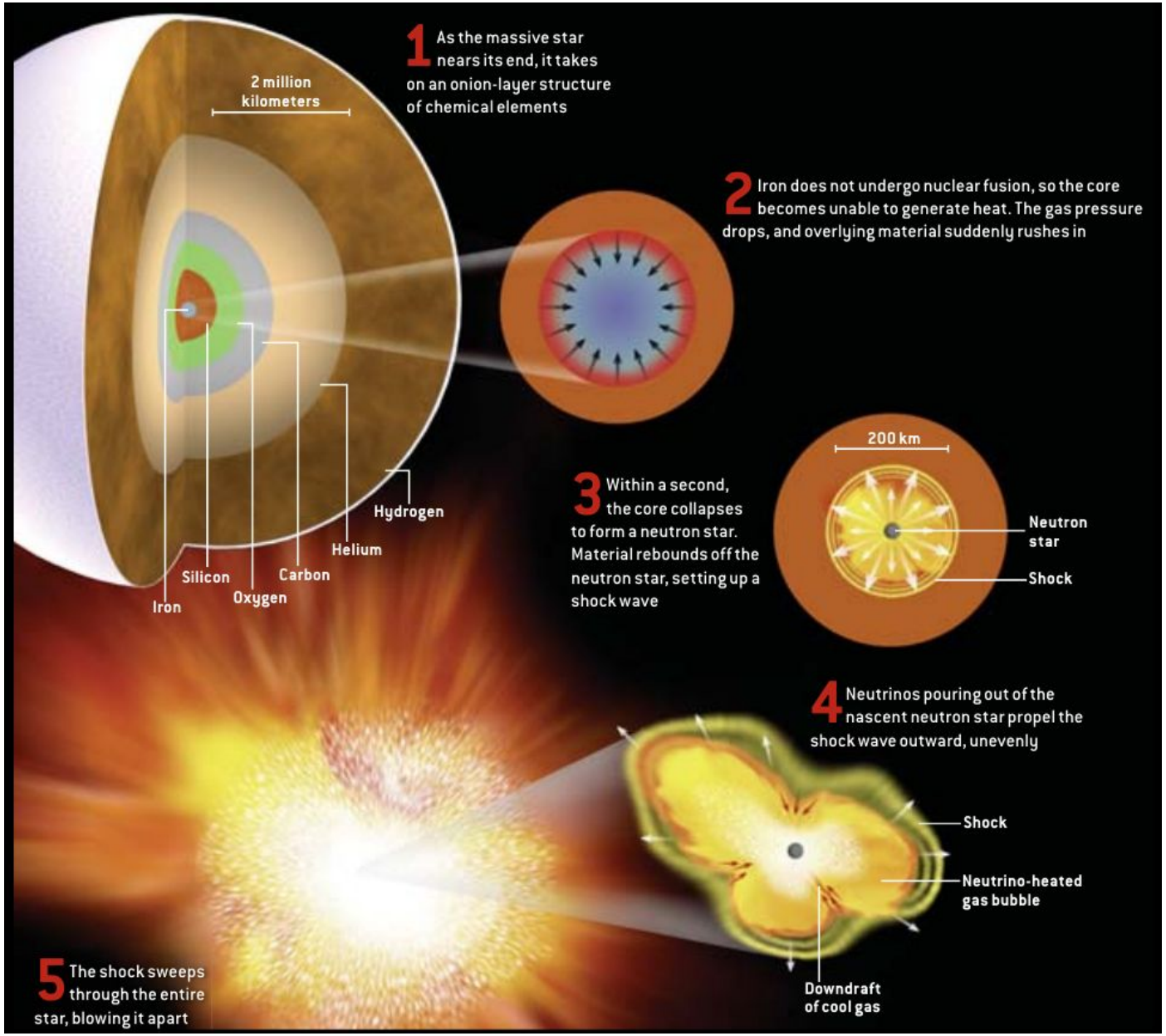


Supernova 1987A

The Path to Explosion



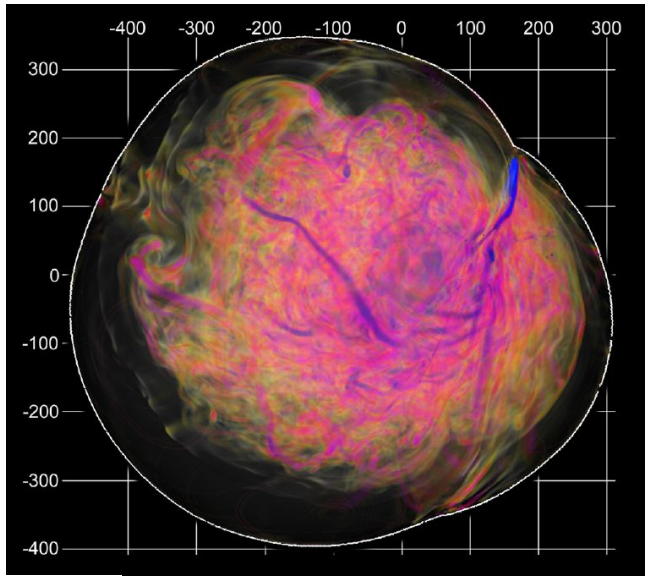
Textbook Supernova



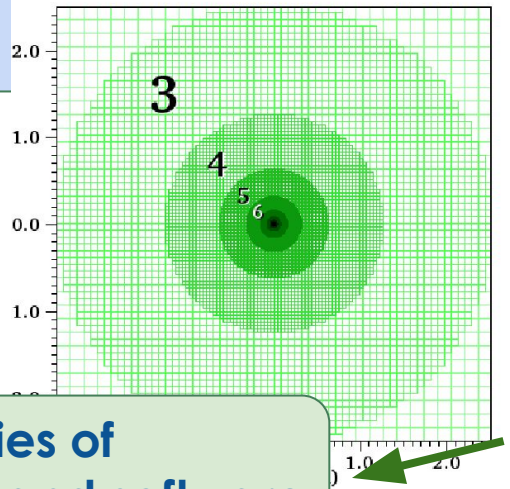
Understanding detail explosion mechanism requires high-fidelity simulations.

Supernova: Multi-Physics & Multi-Scale Problem

- General relativistic gravity
- Compressible Fluid
- Magnetic fields
- Convection
- Nuclear kinetic
- Dense equation of state
- Radiation transport



Daunting software integration tasks



Pushing boundaries of computational hardware and software

- Collapse: ~ 10^6 density increase
- Proto-neutron star: < ~1 km scale
- Outer layer: ~ 10^3 km
- 3D → > 10^9 cells
- 10^{-6}s timestep to 1s total time
- Turbulent cascade
- Magnetorotational instability

Compute power requirements

TEN SECONDS AFTER IGNITION, a thermonuclear flame has almost completed its incineration of a white dwarf star in this recent simulation. Sweeping outward from the deep interior (*cutaway*), the nuclear chain reaction has transformed carbon and oxygen (*lilac, red*) to silicon (*orange*) and iron (*yellow*). Earlier simulations, which were unable to track the turbulent motions, could not explain why stars exploded rather than dying quietly.

How to Blow Up a Star

By Wolfgang Hillebrandt, Hans-Thomas Janka and Ewald Müller

It is not as easy as you would think.

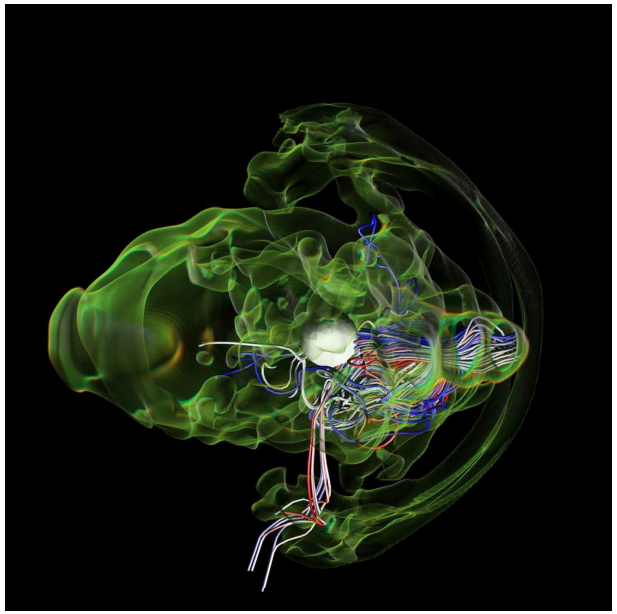
have failed to reproduce these explosions—until recently

On November 11, 1572, Danish astronomer and nobleman Tycho Brahe saw a new star in the constellation Cassiopeia, blazing as bright as Jupiter. In many ways, it was the birth of modern astronomy—a shining disproof of the belief that the heavens were fixed and unchanging. Such "new stars" have not ceased to surprise. Some 400 years later astronomers realized that they briefly outshine billions of ordinary stars and must therefore be spectacular explosions. In 1934 Fritz Zwicky of the California Institute of Technology coined the name "supernovae" for them. Quite apart from being among the most dramatic events known to science, supernovae play a special role in the universe and in the work of astronomers: seeding space with heavy elements, regulating galaxy formation and evolution, even serving as markers of cosmic expansion.

Zwicky and his colleague Walter Baade speculated that the explosive energy comes from gravity. Their idea was that

GenASiS

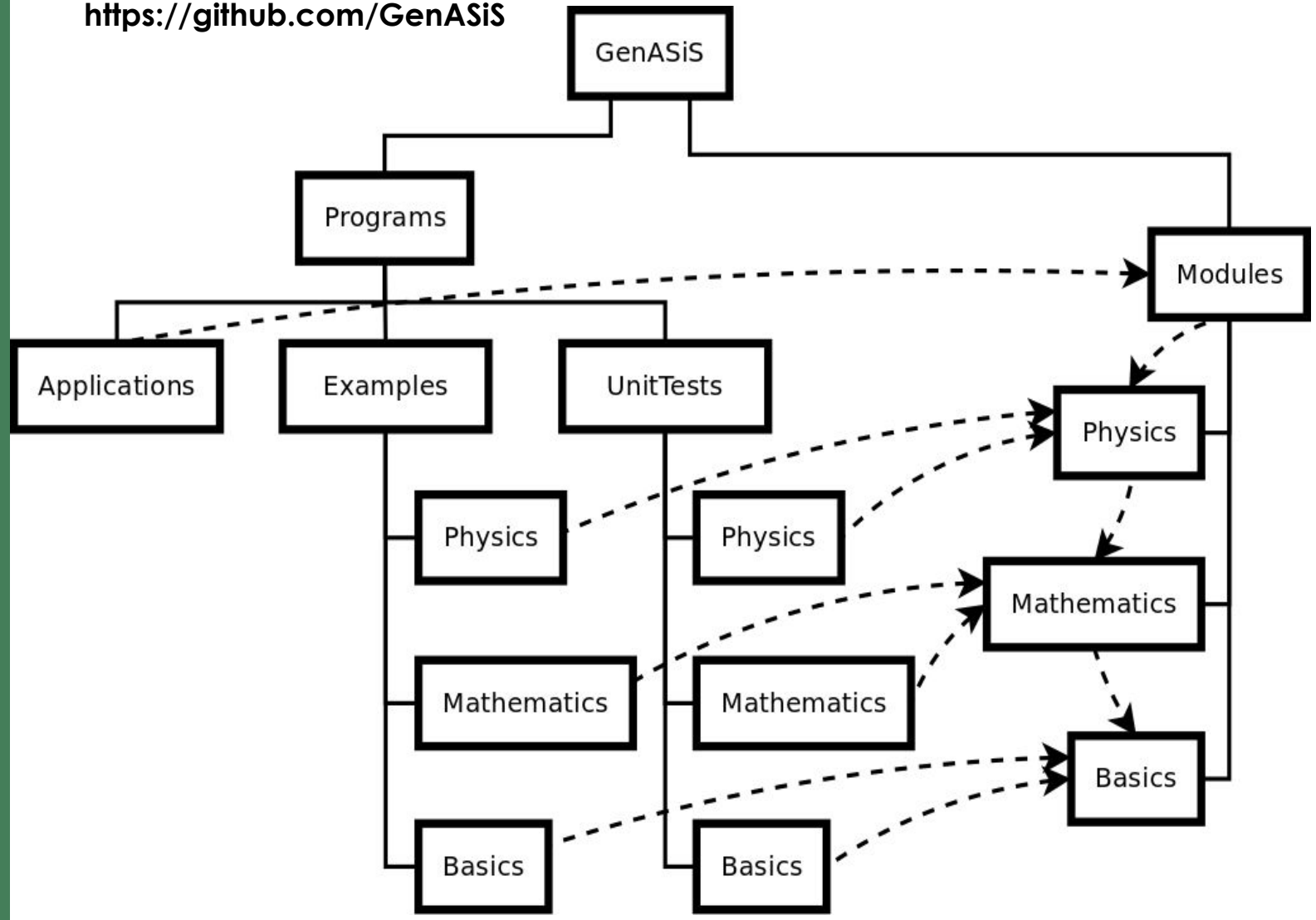
- ***Gen****eral* ***A****strophysics* ***Si****mulation* ***S****ystem*
  - Current target: 3D position space + 1D momentum space the simulations of core-collapse supernovae; Towards 3D + 3D (sustained exascale)
  - Earlier versions have been used to study of fluid instabilities in supernova dynamics, discover exponential magnetic field amplification in progenitor star



- **Code characteristics:**
  - Modern Fortran (mostly F2008, some F2018)
  - Modular, object-oriented design, extensible
  - OpenMP for threading + offloading

GenASiS Structure

https://github.com/GenASiS

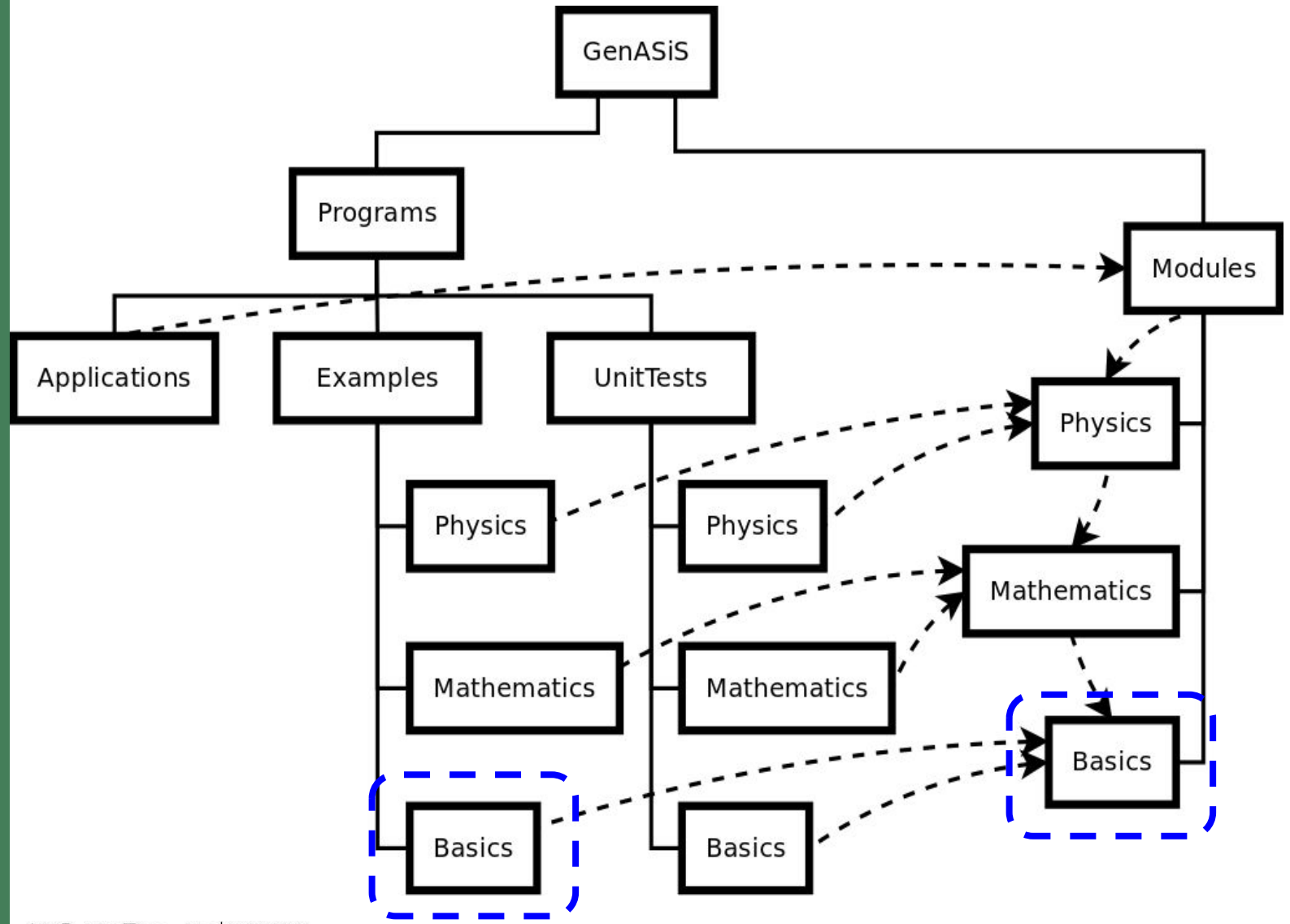


Basics: Utilitarian infrastructure, data management, I/O, Units, Devices, MPI facades

Mathematics: manifolds, solvers (ODE, PDE, elliptic)

Physics: fluid type, equation of states, stress-energy tensors, space-type (newtonian, relativistic, …)

GenASiS Structure



Basics RiemannProblem:

- An example problem using only *Basics classes* with simplified fluid solvers representative of higher-level solvers in (Mathematics, Physics)
- Useful for testings & experimentations with only a handful of computational kernels and simplified mesh (distributed cartesian)

GenASiS Storage Functionality

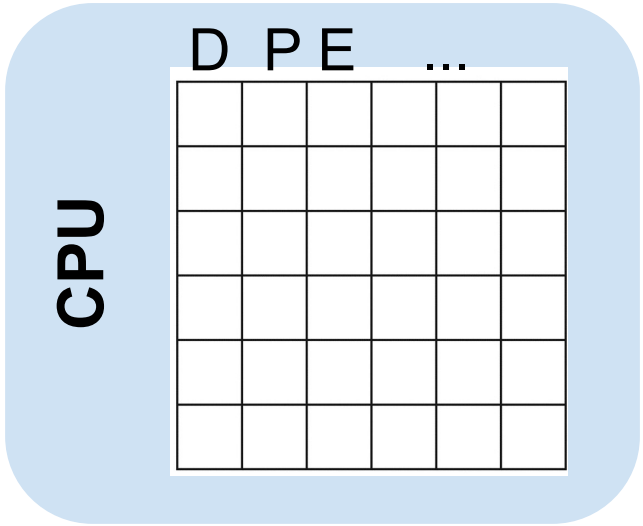
- StorageForm :
  - a class for data and metadata; the 'heart' of data storage facility in GenASiS
  - metadata includes units, variable names (for I/O, visualization)
  - used to group together a set of related physical variables (e.g. Fluid)
  - render more generic and simplified code for I/O, ghost exchange, prolongation & restriction (AMR mesh)

- Data :
  - `StorageForm % Value ( nCells, nVariables )`
  - use as, e.g.
```
Pressure => StorageForm % Value ( :, 1 ),
Density  => StorageForm % Value ( :, 2 )
```

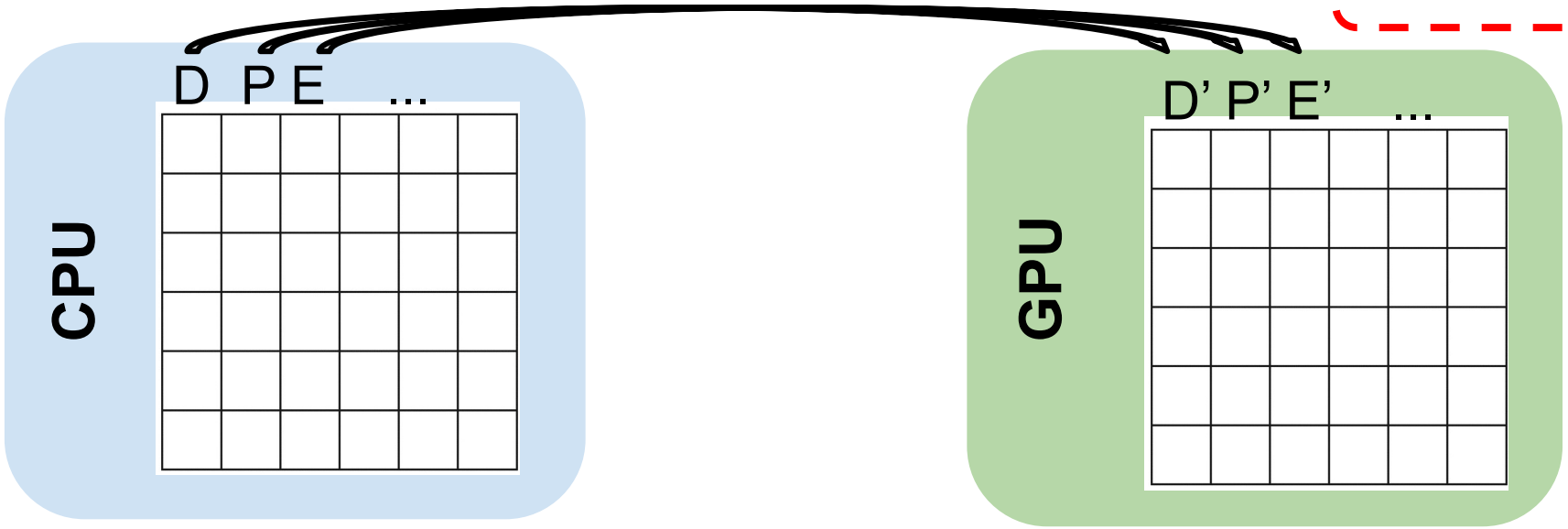
- Methods:
  - `call S % Initialize ( )` ← **allocate** data on **host**
  - `call S % AllocateDevice ( )` ← **allocate** and **associate** data **on GPU**
  - `call S % Update{Device,Host} ( )` ← **transfer** data

Higher-level GenASiS Functionality (2)

- ```
  call StorageForm % Initialize &
         ( Shape = [6, 6], &
           VariableOption = [“Pressure”, “Density”, &
                             “Energy”, … ] )
  ```


  ```

```
call S % AllocateDevice()
```




```
real ( KDR ), dimension ( :, : ), pointer :: Scratch
type ( c_ptr ) :: D_Value

call AllocateDevice ( S % nValues * S % nVariables, D_Value )

call c_f_pointer ( D_Value, Scratch, [ S % nValues, S % nVariables ] )

do iV = 1, S % nVariables
  D_Value = c_loc ( Scratch ( :, iV ) )
  Variable => S % Value ( :, iV )
  call AssociateHost ( D_Value, Variable )
end do
```

Tells OpenMP data location on GPU
→avoid (implicit) allocation & transfer

Lower-Level GenASiS Functionality

- Fortran wrappers to OpenMP APIs
  - `call AllocateDevice(Value, D_Value)`
    → `omp_target_alloc()`

    `call AssociateHost(D_Value, Value)`
    → `omp_target_associate_ptr()`

    `call UpdateDevice(Value, D_Value),`
    `call UpdateHost(Value, D_Value)`
    → `omp_target_memcpy()`

Value : Fortran array
D_Value : type(c_ptr), GPU address

Offloading Computational Kernel

Persistent **allocation** and **association**

```
call F % Initialize &
  ([nCells, nVariables])
call F % AllocateDevice ( )
call F % UpdateDevice ( )
call AddKernel &
  ( F % Value ( :, 1 ),
    F % Value ( :, 2 ), &
    F % Value ( :, 3 ) )
```

```
1  subroutine AddKernel ( A, B, C )
2
3     real ( KDR ), dimension ( : ), intent ( in ) :: A, B
4     real ( KDR ), dimension ( : ), intent ( out ) :: C
5
6     integer ( KDI ) :: i
7
8     !$OMP target teams distribute parallel do schedule ( static, 1 )
9     do i = 1, size ( C )
10       C ( i ) = A ( i ) + B ( i )
11    end do
12    !$OMP end target teams distribute parallel do
13
14 end subroutine AddKernel
```

No implicit data transfer,
no explicit **map()**

Example of Kernel with Pointer Remapping

```
1  subroutine ComputeDifference_X ( V, dV )
2
3    real ( KDR ), dimension ( -1:, -1:, -1: ), &
4      intent ( in ) :: &
5        V
6    real ( KDR ), dimension ( -1:, -1:, -1: ), &
7      intent ( out ) :: &
8        dV
9
10   integer ( KDI ) :: i, j, k
11
12   !$OMP target teams distribute parallel do collapse ( 3 ) schedule ( static, 1 )
13   do k = 1, nZ
14     do j = 1, nY
15       do i = 0, nX + 2
16         dV ( i, j, k ) &
17           = V ( i, j, k ) - V ( i - 1, j, k )
18       end do
19     end do
20   end do
21   !$OMP end target teams distribute parallel do
22
23 end subroutine ComputeDifferences_X
```

```
Caller:

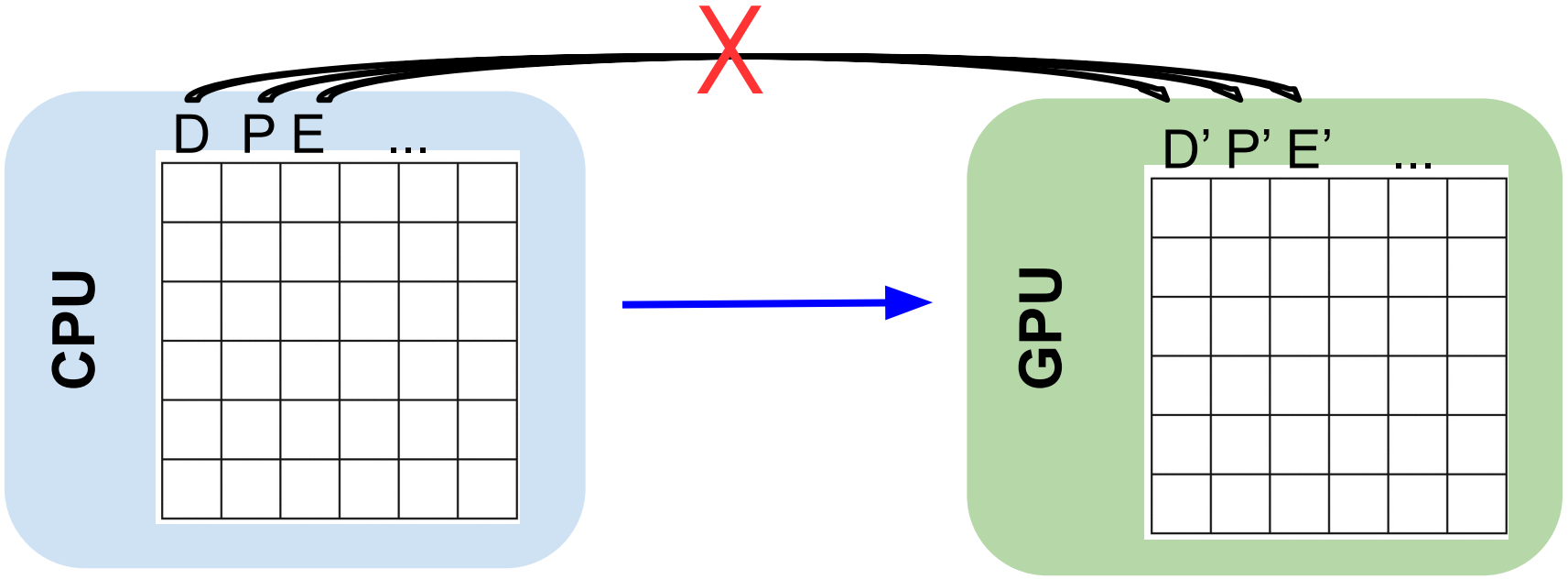
real ( KDR ), dimension ( :, :, : ),
pointer
  :: V, dV

V ( -1:nX+2, -1:nY+2, -1:nZ+2 ) &
  => F % Value ( : , iV )

dV ( -1:nX+2 , -1:nY+2 , -1:nZ+2 ) &
  => dF % Value ( : , iV )

call ComputeDifferences_X ( V, dV )
```

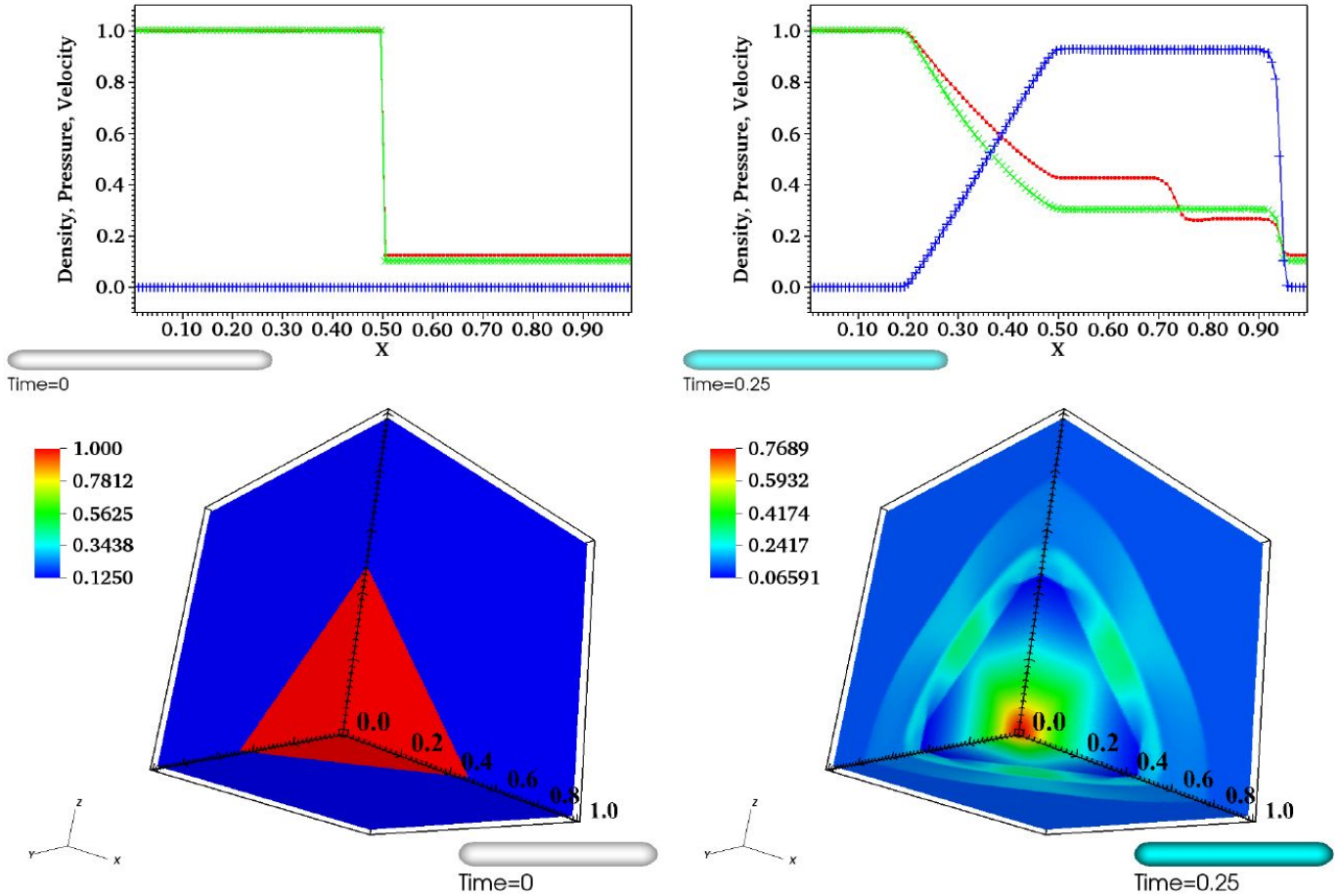
Modify Host - Device Variable Association




```
call S % ReassociateHost &
  ( AssociateVariablesOption = .false. )
```

```
omp_target_disassociate_ptr (...)
omp_target_associate_ptr ( h_ptr, d_ptr, totalSize, …)
```

Basics Riemann Problem

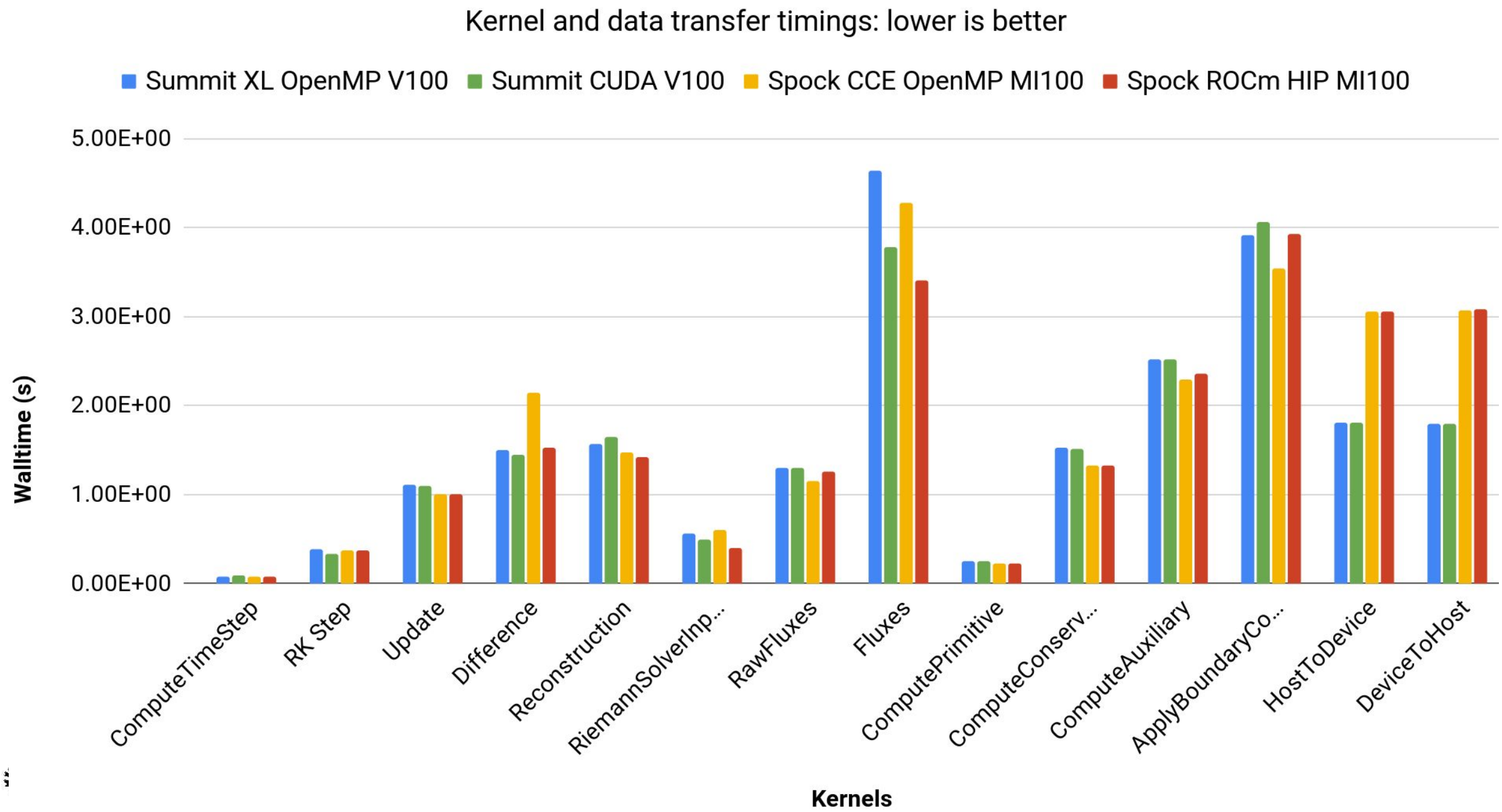


Initial (left) and final (right) density of 1D and 3D RiemannProblem

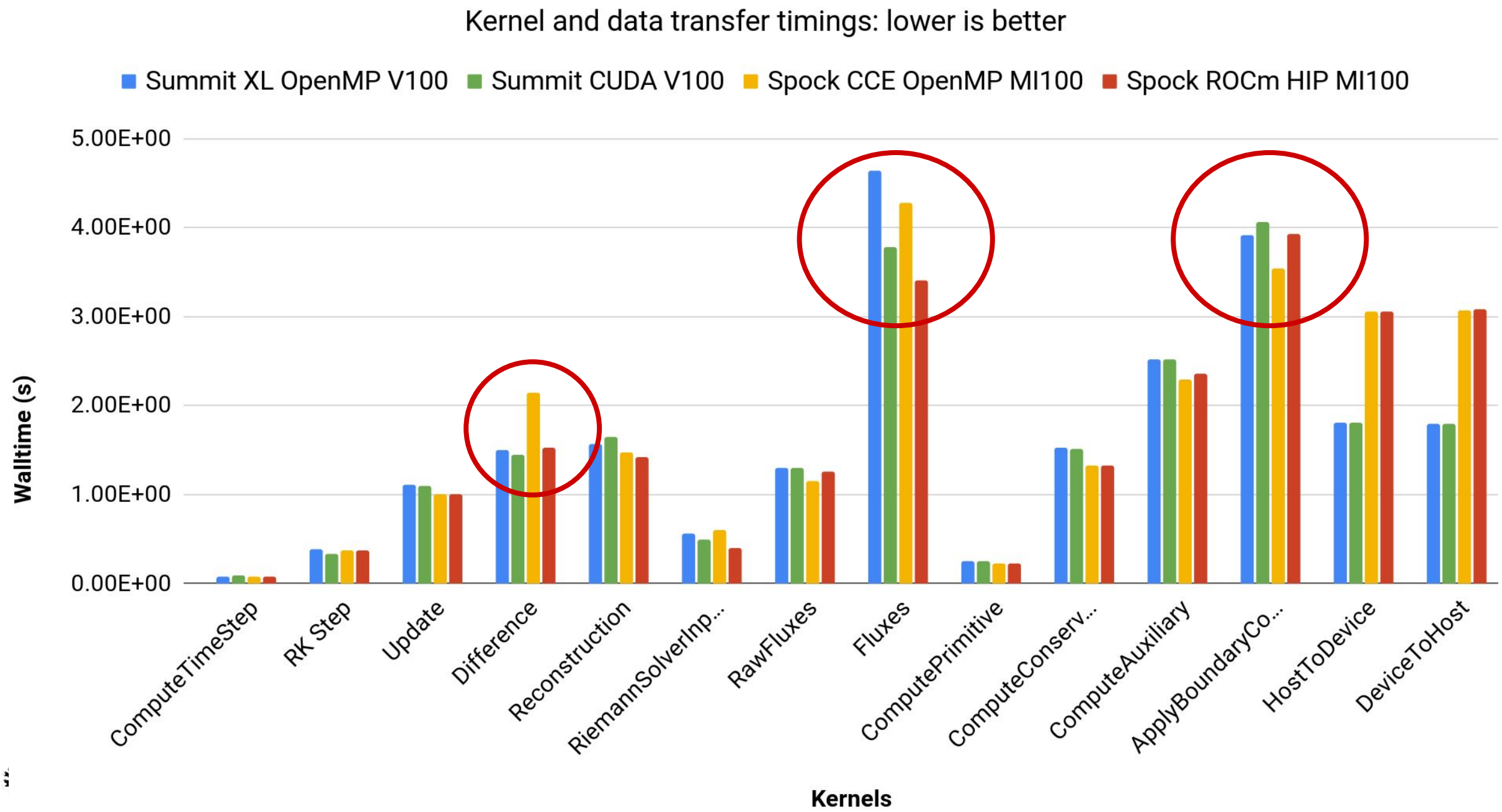
Basics RiemannProblem:

- An example problem using only *Basics classes* with simplified solvers representative of higher-level solvers in (Mathematics, Physics)
- Useful for testings & experimentations with only a handful of computational kernels and simplified mesh (distributed cartesian)
- Similar solvers are used for the explicit part of radiation transport in an IMEX scheme
- Kernels have also been ported to CUDA / HIP for comparisons

GenASiS Basics: RiemannProblem 3D, 256^3 per GPU, 1 process, 50 cycles



GenASiS Basics: RiemannProblem 3D, 256^3 per GPU, 1 process, 50 cycles



Future Work (1): Metadirective

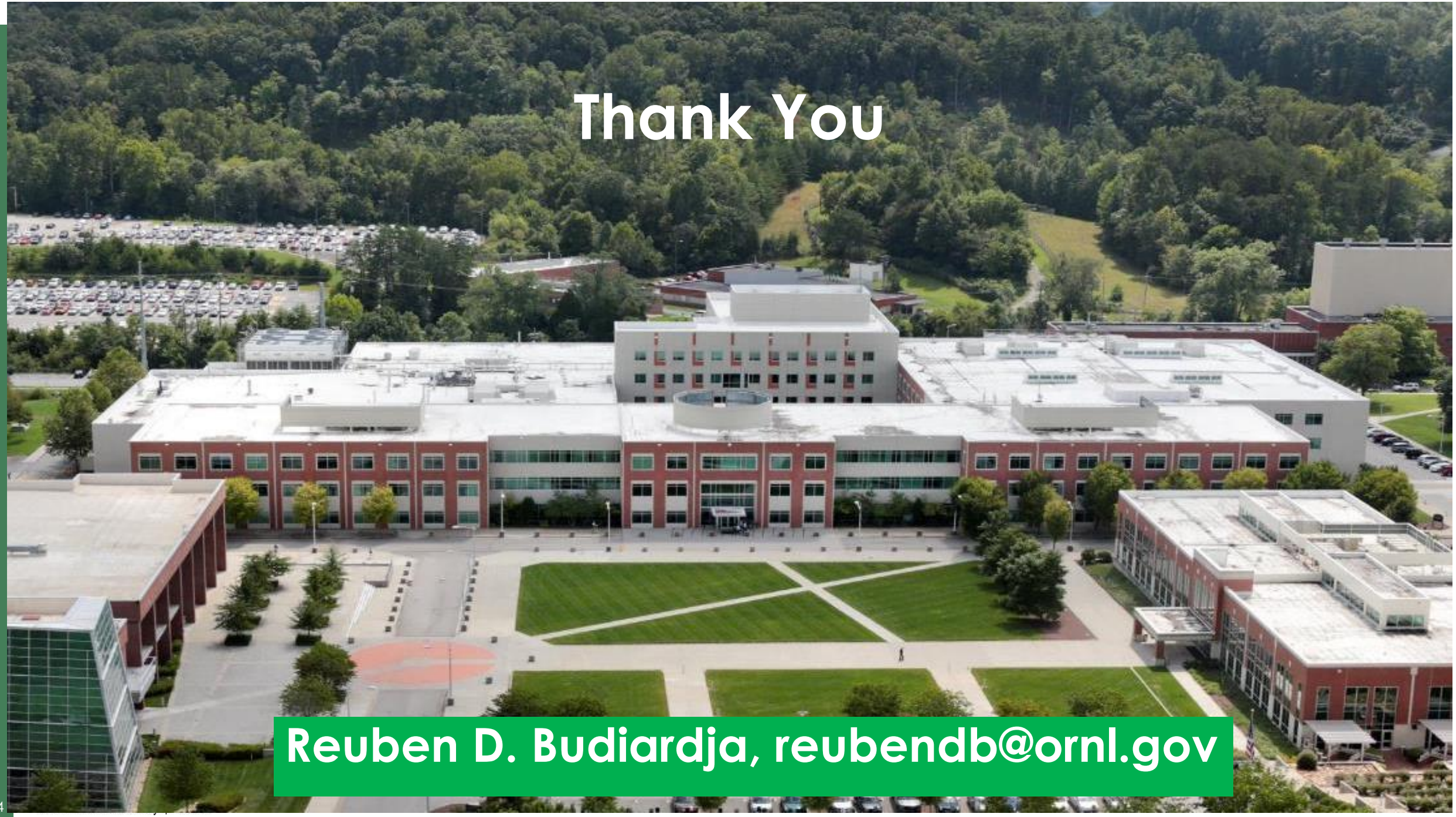
- Use case: evolve fluid on host + radiation on GPU + load balancing
- Currently code duplication needed for compiler to generate both device and host versions.
- Better: metadirective with user-selector and dynamic condition

```
if ( UseDevice ) then
  !$OMP  target teams distribute parallel do &
  !$OMP& schedule ( static, 1 ) private ( KE )
  do iV = 1, size ( E )
    [kernel code ...]
  end do
  !$OMP end target teams distribute parallel do
else
  !$OMP  parallel do &
  !$OMP& schedule ( runtime ) private ( KE )
  do iV = 1, size ( E )
    [duplicated kernel code ...]
  end do
  !$OMP end parallel do
end if
```

```
!$OMP begin metadirective &
!$OMP when ( user = { condition ( UseDevice ) } &
             : target teams distribute parallel do &
!$OMP   default: parallel do
!$OMP private ( KE )
do iV = 1, size ( E )
    [kernel code ...]
end do
!$OMP end metadirective
```

Future Work (2): Remove Compatibility Interfaces

- Currently uses [cuda/hip]HostMalloc() to allocate page-locked (pinned) memory → OpenMP 5 allocator
- Some OpenMP 4.5 library routine is in C only. Fixed in OpenMP 5.


Thank You
Reuben D. Budiardja, reubendb@ornl.gov